

# Babai's Nearest Plane Algorithm

Eric Ren, Sage Binder, and Katherine Kosaian

February 6, 2026

## Abstract

$\gamma$ -CVP is the problem of finding a vector in  $L$  that is within  $\gamma$  times the closest possible to  $t$ , where  $L$  is a lattice and  $t$  is a target vector. If the basis for  $L$  is LLL-reduced, Babai's Closest Hyperplane algorithm solves  $\gamma$ -CVP for  $\gamma = 2^{n/2}$ , where  $n$  is the dimension of the lattice  $L$ , in time polynomial in  $n$ . This session formalizes said algorithm, using the AFP formalization of LLL [2, 1] and adapting a proof of correctness from the lecture notes of Stephens-Davidowitz [4].

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Locale setup for Babai</b>	<b>4</b>
<b>3</b>	<b>Coordinates</b>	<b>6</b>
<b>4</b>	<b>Lattice Lemmas</b>	<b>16</b>
<b>5</b>	<b>Lemmas on closest distance</b>	<b>19</b>
<b>6</b>	<b>More linear algebra lemmas</b>	<b>24</b>
<b>7</b>	<b>Coord-Invariance</b>	<b>31</b>
<b>8</b>	<b>Bound on distance to target, with <math>\epsilon</math> factor.</b>	<b>39</b>

## 1 Introduction

The (exact) *closest vector problem* (CVP) is the problem of finding the closest vector within a lattice  $L$  to a target vector  $t$ . This is equivalent to finding the shortest vector in the *lattice coset*  $L - t := \{l - t : l \in L\}$ . There is a corresponding family of weaker problems,  $\gamma$ -CVP (where  $\gamma$  is some real parameter), where one needs only find a vector in  $L - t$  whose length is at most  $\gamma$  times the shortest possible. Through a reduction to the *shortest*

*vector problem* [4], solutions to these problems may be used to factor rational polynomials. This problem is therefore of cryptographic interest.

Although exact CVP (or 1-CVP) is NP-Complete [3], Babai's Nearest Plane Algorithm solves  $2^{n/2}$ -CVP, where  $n$  is the dimension of  $L$ , in polynomial time, provided that  $L$  is presented using an LLL-reduced basis with parameter  $\alpha = 4/3$ . The proof in this document is mostly a straightforward algebraicization of the proof in Stephens-Davidowitz' lecture notes. It makes use of the coordinate systems defined by the original basis (denoted  $\beta$ ) and the Gram-Schmidt orthogonalization of that basis (denoted  $\tilde{\beta}$ ). Let  $[u]_\beta$  denote the representation of a vector  $u$  under  $\beta$ , with coordinates  $[u]_\beta^j$ ;  $j = 1, \dots, n$  (likewise for  $\tilde{\beta}$ ). Also, let  $s_i$  denote the output of the algorithm after step  $i$  and let  $d$  be the shortest lattice coset vector, as witnessed by the vector  $v$ . The proof works by analysing the coordinates of  $[s_n]_{\tilde{\beta}}$ , showing that all are at most  $1/2$  and that some later coordinates are exactly those of  $[v]_{\tilde{\beta}}$ .

The algorithm modifies coordinate  $n - i$  in both bases for the last time in step  $i$  (formalized in lemma `coord_invariance`), during which both coordinates are decreased below  $1/2$  (formalized in lemma `small_coord`). Combined, these facts imply that the output  $s_n$  has  $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$  for all indices  $j$ .

Since  $\tilde{\beta}$  is orthogonal, we have

$$\|s_n\|^2 = \sum_{i=1}^n \left( [s_n]_{\tilde{\beta}}^i \|\tilde{\beta}_i\| \right)^2, \quad (1)$$

so the preceding coordinate bounds  $\|s_n\|^2$  by  $\frac{1}{4} \sum_{i=1}^n \|\tilde{\beta}_i\|^2$ . If the  $\tilde{\beta}_i$  are all short compared to  $d$ , this bound suffices. In fact, if there is any short vector  $\tilde{\beta}_I$  in  $\tilde{\beta}$  then because  $\beta$  is LLL-reduced, any vector preceding  $\tilde{\beta}_I$  in  $\tilde{\beta}$  will not be much longer. This bounds the first  $I$  terms in Equation 1. By selecting  $I$  maximal, we may assume that  $\tilde{\beta}$  ends in a series of  $n - I$  long vectors. In this case it can be shown  $[v]_{\tilde{\beta}}^j$  and  $[s_n]_{\tilde{\beta}}^j$  differ by an integral amount for  $j = I + 1, \dots, n$ . Therefore, if  $[v]_{\tilde{\beta}}^j$  and  $[s_n]_{\tilde{\beta}}^j$  differ at all, they differ by at least 1, which would mean  $\left|[v]_{\tilde{\beta}}^j\right| \geq 1/2$ , since  $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$ . This would force  $v$  to be longer than  $d$ , a contradiction. So  $[v]_{\tilde{\beta}}^j = [s_n]_{\tilde{\beta}}^j$  for  $j = I + 1, \dots, n$ , which gives a tighter bound on the last  $n - I$  terms in equation 1.

Precisely, let  $I$  denote  $\max\{i : \|\tilde{\beta}_i\| \leq 2d\}$ , meaning for all indices  $j > I$ ,  $\|\tilde{\beta}_j\| > 2d$ . Now, for all  $j > I$ ,  $d^2 = \|v\|^2 \geq ([v]_{\tilde{\beta}}^j)^2 \|\tilde{\beta}_j\|^2 > ([v]_{\tilde{\beta}}^j)^2 \cdot 4d^2$ , meaning  $1/4 > (\tilde{\beta}^j)^2$ , or  $1/2 > \left|[v]_{\tilde{\beta}}^j\right|$ . Since  $\left|[s_j]_{\tilde{\beta}}^j\right| \leq 1/2$  from the previous section,  $\left|[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j\right| < 1$ . Using properties of the change-of-basis between  $\beta, \tilde{\beta}$  formalized in the LLL AFP session, we show that  $[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j =$

$[v]_\beta^j - [s_j]_\beta^j = [v - s_j]_\beta^j$ , so that  $\left| [v - s_j]_\beta^j \right| < 1$ . But since  $v - s_j$  lies in the lattice,  $[v - s_j]_\beta^j$  is integral, so  $\left| [v - s_j]_\beta^j \right| = 0$ , meaning  $[v]_\beta^j = [s_j]_\beta^j$ . Lemma `coord_invariance` gives that  $[v]_\beta^j = [s_j]_\beta^j = [s_n]_\beta^j$ . This is formalized by lemma `correct_coord`.

Now  $\|s_n\|^2 = \sum_{i=1}^n ([s_n]_{\tilde{\beta}}^i \|\tilde{\beta}_i\|)^2$ , since  $\tilde{\beta}$  is orthogonal. Splitting the sum around  $I$  equates this to  $\sum_{i=1}^I ([s_n]_{\tilde{\beta}}^i)^2 + \sum_{i=I+1}^n ([s_n]_{\tilde{\beta}}^i)^2$ . Lemma `small_coord` bounds the terms in the first sum by  $\|\tilde{\beta}_i\|^2/4$ , while lemma `correct_coord` bounds the terms in the second sum by  $d^2$ , giving  $\|s_n\|^2 \leq (n - I)d^2 + \sum_{i=1}^I \|\tilde{\beta}_i\|^2/4$ . If  $\beta$  is LLL-reduced with parameter  $\alpha$ ,  $\|\tilde{\beta}_i\|^2 \leq \alpha^I \|\tilde{\beta}_I\|^2$  for all  $i \leq I$ , which, by the definition of  $I$ , is at most  $4d^2$ . So  $\|s_n\|^2 \leq ((n - I) + I\alpha^I)d^2 \leq n\alpha^n d^2$ . The standard choice of  $\alpha = 4/3$  gives  $\|s_n\|^2 \leq 2^n d^2$ . All of this is formalized in the final section, which culminates in the main theorem.

To avoid having to prove that a shortest vector exists, we use the definition  $\inf\{\|u - t\| : u \in L\}$  for  $d$  instead of  $\min\{\|u - t\| : u \in L\}$  and rephrase the arguments above to allow  $\|v\|$  to exceed  $d$  by a small constant factor  $\epsilon$ . This workaround and its details are contained in the section on the closest distance and negligibly change the rest of the proof.

**theory** *Babai-Algorithm*

**imports**

*LLL-Basis-Reduction.LLL-Impl*

*HOL.Archimedean-Field*

*HOL-Analysis.Inner-Product*

**begin**

**fun** *calculate-c*:: *rat vec*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *nat*  $\Rightarrow$  *int* **where**

*calculate-c s L1 n = round ((s \* (L1!((dim-vec s) - n))) / (sq-norm-vec (L1!((dim-vec s) - n))))*

**fun** *update-s*:: *rat vec*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *nat*  $\Rightarrow$  *rat vec* **where**

*update-s sn M Mt n = ((rat-of-int (calculate-c sn Mt n))  $\cdot_v$  M!((dim-vec sn) - n))*

**fun** *babai-help*:: *rat vec*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *nat*  $\Rightarrow$  *rat vec* **where**

*babai-help s M Mt 0 = s |*

*babai-help s M Mt (Suc n) = (let B = (babai-help s M Mt n) in B - (update-s B M Mt (Suc n)))*

This assumes an LLL-reduced input and outputs a short vector of the form  $v + t$ , with  $v \in L$

**fun** *babai-of-LLL* :: *rat vec*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  *rat vec* **where**  
*babai-of-LLL* *s M* = *babai-help* *s M* (*gram-schmidt* (*dim-vec s*) *M*) (*dim-vec s*)

This begins with a non-reduced basis and outputs a vector  $v$  in  $L$  which is close to  $t$

**fun** *full-babai* :: *int vec list*  $\Rightarrow$  *rat vec*  $\Rightarrow$  *rat*  $\Rightarrow$  *int vec* **where**  
*full-babai* *fs target*  $\alpha$  =  
*map-vec int-of-rat*  
(*babai-of-LLL*  
(*uminus target*)  
(*LLL.RAT* (*LLL-Impl.reduce-basis*  $\alpha$  *fs*))  
+ *target*)

**end**

**theory** *Babai-Correct*

**imports** *Babai-Algorithm*

*LLL-Basis-Reduction.LLL-Impl*

*Jordan-Normal-Form.DL-Rank*

*BenOr-Kozen-Reif.More-Matrix*

**begin**

This theory contains the proof of correctness of the algorithm. The main theorem is “theorem babai-correct”, under the locale “babai-with-assms”. To use the theorem, one needs to show that lattice, the vectors in the lattice basis, and the target vector all have the same dimension, that the lattice basis vectors are linearly independent, and that the lattice basis is LLL-weakly-reduced for some parameter  $\alpha \geq 4/3$ .

## 2 Locale setup for Babai

**locale** *babai* =  
**fixes** *M* :: *int vec list*  
**fixes** *t* :: *rat vec*  
**fixes**  $\alpha$  :: *rat*  
**assumes** *length-M*: *length M* = *dim-vec t*  
**begin**

**abbreviation** *n* **where**  $n \equiv \text{length } M$

**sublocale** *LLL n n M*  $\alpha$  .

**abbreviation** *coset*::*rat vec set* **where**  $\text{coset} \equiv \{(map-vec \text{rat-of-int } x) - t \mid x. x \in L\}$

**abbreviation** *Mt* **where**  $Mt \equiv \text{gram-schmidt } n \text{ (RAT } M)$

**definition** *s* :: *nat*  $\Rightarrow$  *rat vec* **where**

$s \ i = \text{babai-help } (uminus \ t) \ (\text{RAT } M) \ Mt \ i$

**definition** *closest-distance-sq*::*real* **where**

$\text{closest-distance-sq} = \text{Inf } \{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\}$

**end**

Locale setup with additional assumptions required for main theorem. Contains an arbitrary extra constant  $\epsilon$  which appears in the final bound in this locale, giving a theorem quantified over all  $\epsilon > 1$ . The next locale, “babai-with-assms,” uses this theorem to prove a theorem without  $\epsilon$ .

```
locale babai-with-assms- $\epsilon$  = babai +
  fixes mat-M mat-M-inv:: rat mat
  fixes  $\epsilon$  :: real
  assumes basis: lin-indep M
  defines mat-M  $\equiv$  mat-of-cols n (RAT M)
  defines mat-M-inv  $\equiv$ 
    (if (invertible-mat mat-M) then SOME B. (inverts-mat B mat-M)  $\wedge$  (inverts-mat
mat-M B) else (0m n n))
  assumes inv: invertible-mat mat-M
  assumes reduced: weakly-reduced M n
  assumes non-trivial: 0 < n
  assumes alpha:  $\alpha \geq 4/3$ 
  assumes  $\epsilon$ :  $\epsilon > 1$ 
begin
```

```
lemma dim-vecs-in-M:
  shows  $\forall v \in \text{set } M. \text{dim-vec } v = \text{length } M$ 
  using basis unfolding gs.lin-indpt-list-def by force
```

```
lemma inv1:mat-M * mat-M-inv = 1m n
proof –
  have dim-m:dim-row mat-M = n using dim-vecs-in-M unfolding mat-M-def
by fastforce
  then have inverts-mat mat-M mat-M-inv using inv
  unfolding mat-M-inv-def
  by (smt (verit, ccfv-SIG) invertible-mat-def some-eq-imp)
  then show ?thesis using dim-m unfolding inverts-mat-def by argo
qed
```

```
lemma inv2:mat-M-inv * mat-M = 1m n
proof –
  have dim-m:dim-col mat-M = n unfolding mat-M-def by fastforce
  have inverts-mat mat-M-inv mat-M using inv
  unfolding mat-M-inv-def
  by (smt (verit, ccfv-SIG) invertible-mat-def some-eq-imp)
  then have inv:mat-M-inv * mat-M = 1m (dim-row mat-M-inv)
  unfolding inverts-mat-def by blast
  then have dim-n:dim-col (1m (dim-row mat-M-inv)) = n
  using dim-m index-mult-mat(3)[of mat-M-inv mat-M] by fastforce
  have (dim-row mat-M-inv) = n
  proof(rule ccontr)
  assume (dim-row mat-M-inv)  $\neq$  n
  then have dim-col (1m (dim-row mat-M-inv))  $\neq$  n
```

```

    by auto
    then show False using dim-n by blast
qed
then show ?thesis using inv by argo
qed

```

**sublocale** *rats*: *vec-module TYPE(rat) n*.

**lemma** *M-dim*:  $\dim\text{-row } \text{mat-}M = n \ \dim\text{-col } \text{mat-}M = n$   
**apply** (*metis index-mult-mat(2) index-one-mat(2) inv1*)  
**by** (*metis index-mult-mat(3) index-one-mat(3) inv2*)

**lemma** *M-inv-dim*:  $\dim\text{-row } \text{mat-}M\text{-inv} = n \ \dim\text{-col } \text{mat-}M\text{-inv} = n$   
**apply** (*metis M-dim(1) index-mult-mat(2) inv1 inv2*)  
**by** (*metis index-mult-mat(3) index-one-mat(3) inv1*)

**lemma** *babai-to-help*:  
**shows**  $s \ n = \text{babai-of-LLL } (\text{uminus } t) \ (\text{RAT } M)$   
**using** *babai-def babai.s-def babai-of-LLL.simps babai-axioms by force*

### 3 Coordinates

This section sets up the use of the lattice basis and its GS orthogonalization as coordinate systems and some properties of that coordinate system. The important lemma here is *coord-invariance*, which shows that after step *i* of the algorithm, all coordinates (in both systems) after *n-i* are invariant.

**definition** *lattice-coord* ::  $\text{rat } \text{vec} \Rightarrow \text{rat } \text{vec}$   
**where** *lattice-coord* *a* =  $\text{mat-}M\text{-inv } *_v \ a$

**lemma** *dim-preserve-lattice-coord*:  
**fixes** *v*:: $\text{rat } \text{vec}$   
**assumes**  $\dim\text{-vec } v = n$   
**shows**  $\dim\text{-vec } (\text{lattice-coord } v) = n$  **unfolding** *lattice-coord-def mat-M-inv-def*  
**using** *M-inv-dim*  
**by** (*simp add: mat-M-inv-def*)

**lemma** *vec-to-col*:  
**assumes**  $i < n$   
**shows**  $(\text{RAT } M)!i = \text{col } \text{mat-}M \ i$   
**unfolding** *mat-M-def*  
**by** (*metis babai-with-assms-ε-axioms babai-with-assms-ε-axioms-def babai-with-assms-ε-def M-dim(2)*)  
*assms cols-mat-of-cols cols-nth gs.lin-indpt-list-def mat-M-def*)

**lemma** *unit*:  
**assumes**  $i < n$   
**shows**  $\text{lattice-coord } ((\text{RAT } M)!i) = \text{unit-vec } n \ i$   
**using** *assms inv2 unfolding lattice-coord-def*

by (metis *M-dim(1) M-dim(2) M-inv-dim(2) carrier-matI col-mult2 col-one vec-to-col*)

lemma *linear*:

fixes *i::nat*  
fixes *v1::rat vec*  
and *v2:: rat vec*  
and *q:: rat*  
assumes *dim-vec v1 = n*  
assumes *dim-2: dim-vec v2 = n*  
assumes  $0 \leq i$   
assumes *dim-i: i < n*  
shows (lattice-coord ( $v1 + (q \cdot_v v2)$ )) $\$i =$  (lattice-coord *v1*) $\$i + q * ($ lattice-coord *v2*) $\$i$ )  
using *assms*  
proof(–)  
have *linear-vec*:(lattice-coord ( $v1 + (q \cdot_v v2)$ )) = (lattice-coord *v1*) + *q*  $\cdot_v$  ((lattice-coord *v2*))  
unfolding *lattice-coord-def*  
by (metis (*mono-tags, opaque-lifting*) *M-inv-dim(2) assms(1) assms(2) carrier-mat-triv carrier-vec-dim-vec mult-add-distrib-mat-vec mult-mat-vec smult-carrier-vec*)  
then have *2*: (lattice-coord ( $v1 + (q \cdot_v v2)$ )) $\$i =$  ((lattice-coord *v1*) + *q*  $\cdot_v$  ((lattice-coord *v2*))) $\$i$  by *auto*  
also have *dim-v2*: *dim-vec* (lattice-coord *v2*) = *n* using *dim-preserve-lattice-coord dim-2* by *blast*  
then have *i-in-range*:  $i < \text{dim-vec } (q \cdot_v (\text{lattice-coord } v2))$  using *dim-v2 dim-i* by *simp*  
also have *3*:((lattice-coord *v1*) + *q*  $\cdot_v$  ((lattice-coord *v2*))) $\$i =$  (lattice-coord *v1*) $\$i + (q \cdot_v$  (lattice-coord *v2*)) $\$i$   
using *i-in-range* by *simp*  
also have *4*: ( $q \cdot_v$  (lattice-coord *v2*)) $\$i = q * ($ lattice-coord *v2*) $\$i$  using *i-in-range* by *simp*  
thus *?thesis* unfolding *vec-def* using *linear-vec 2 3 4* by *simp*  
qed

lemma *sub-s*:

fixes *i::nat*  
assumes  $0 \leq i$   
assumes  $i < n$   
shows  $s (\text{Suc } i) = (s \ i) -$   
( $(\text{rat-of-int } (\text{calculate-c } (s \ i) \text{ Mt } (\text{Suc } i)) \cdot_v (\text{RAT } M))!((\text{dim-vec } (s \ i)) - (\text{Suc } i)))$ )  
using *assms babai-help.simps[of -t RAT M Mt]* unfolding *s-def*  
by (*metis update-s.simps*)

lemma *M-locale-1*:

shows *gram-schmidt-fs-Rn n (RAT M)*  
by (*smt (verit) M-dim(1) M-dim(2) carrier-dim-vec dim-col gram-schmidt-fs-Rn.intro*)

*in-set-conv-nth*  
 $mat-M-def\ mat-of-cols-carrier(3)\ subset-code(1)\ vec-to-col$

**lemma** *M-locale-2*:

**shows** *gram-schmidt-fs-lin-indpt*  $n$  (*RAT*  $M$ )  
**using** *basis M-locale-1 gram-schmidt-fs-lin-indpt.intro*[*of*  $n$  (*RAT*  $M$ )] **unfolding**  
*gs.lin-indpt-list-def*  
**using** *gram-schmidt-fs-lin-indpt-axioms.intro* **by** *blast*

**lemma** *more-dim*:  $length\ (RAT\ M) = n$   
**by** *simp*

**lemma** *Mt-gso-connect*:

**fixes**  $j :: nat$   
**assumes**  $j < n$   
**shows**  $Mt!j = gs.gso\ j$   
**proof**(-)  
**have**  $Mt = map\ gs.gso[0..<n]$   
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of*  $n$  (*RAT*  $M$ )]  
**by** *fastforce*  
**then show** *?thesis*  
**using** *assms*  
**by** *simp*

**qed**

**lemma** *access-index-M-dim*:

**assumes**  $0 \leq i$   
**assumes**  $i < n$   
**shows**  $dim-vec\ (map\ of-int-hom.vec-hom\ M\ !\ i) = n$   
**using** *assms dim-vecs-in-M*  
**by** *auto*

**lemma** *s-dim*:

**fixes**  $i::nat$   
**assumes**  $i \leq n$   
**shows**  $dim-vec\ (s\ i) = n \wedge (s\ i) \in carrier-vec\ n$   
**using** *assms*  
**proof**(*induct*  $i$ )  
**case**  $0$   
**have** *unfold1*:  $s\ 0 = babai-help\ (uminus\ t)\ (RAT\ M)\ Mt\ 0$  **unfolding** *s-def* **by**  
*simp*  
**also have** *unfold2*:  $babai-help\ (uminus\ t)\ (RAT\ M)\ Mt\ 0 = uminus\ t$  **unfolding**  
*babai-help.simps* **by** *simp*  
**also have** *unfold3*:  $s\ 0 = uminus\ t$  **using** *unfold1* *unfold2* **by** *simp*  
**also have** *dim-eq*:  $dim-vec\ (s\ 0) = dim-vec\ (uminus\ t)$  **using** *unfold3* **by** *simp*  
**moreover have** *dim-minus*:  $dim-vec\ (uminus\ t) = n$  **by** (*metis index-uminus-vec(2)*  
*length-M*)  
**then have**  $dim-vec\ (s\ 0) = n$

```

    using dim-eq dim-minus
    by simp
  then have  $(s\ 0) \in \text{carrier-vec } n$ 
    using carrier-vecI[of  $(s\ 0)\ n$ ]
    by simp
  then show ?case
    by simp
next
case (Suc i)
  then have  $\text{leq: } i \leq n$  by linarith
  have  $\text{sub:s } (\text{Suc } i) = (s\ i) - ((\text{rat-of-int } (\text{calculate-c } (s\ i)\ \text{Mt } (\text{Suc } i)) ) ) \cdot_v$ 
 $(\text{RAT } M)!((\text{dim-vec } (s\ i)) - (\text{Suc } i))$ 
    using sub-s Suc
    by auto
  moreover have  $\text{prev-s-dim:}(s\ i) \in \text{carrier-vec } n$ 
    using Suc
    by simp
  moreover have  $\text{dim-vec } (s\ i) = n$ 
    using Suc
    by simp
  then have  $0 \leq (\text{dim-vec } (s\ i)) - (\text{Suc } i) \wedge (\text{dim-vec } (s\ i)) - (\text{Suc } i) < n$ 
    using Suc
    by linarith
  then have  $\text{dim-m: } (\text{dim-vec } ((\text{RAT } M)!((\text{dim-vec } (s\ i)) - (\text{Suc } i)))) = n$ 
    using access-index-M-dim[of  $(\text{dim-vec } (s\ i)) - (\text{Suc } i)$ ]
    by simp
  then have  $\text{dim-qm:dim-vec } ((\text{rat-of-int } (\text{calculate-c } (s\ i)\ \text{Mt } (\text{Suc } i)) ) ) \cdot_v$ 
 $(\text{RAT } M)!((\text{dim-vec } (s\ i)) - (\text{Suc } i)) = n$ 
    by simp
  then have  $\text{final-dim:dim-vec } ((s\ i) -$ 
 $((\text{rat-of-int } (\text{calculate-c } (s\ i)\ \text{Mt } (\text{Suc } i)) ) ) \cdot_v (\text{RAT } M)!((\text{dim-vec } (s\ i)) -$ 
 $(\text{Suc } i)))) = n$ 
    using index-minus-vec(2) prev-s-dim dim-qm
    by metis
  show ?case
    using final-dim sub carrier-vecI[of  $s\ i\ n$ ]
    by (metis carrier-vec-dim-vec)
qed

```

**lemma** *dim-vecs-in-Mt:*

```

  fixes  $i::\text{nat}$ 
  assumes  $i < n$ 
  shows  $\text{dim-vec } (\text{Mt } i) = n$ 
  using Mt-gso-connect[of  $i$ ] M-locale-1 assms gram-schmidt-fs-Rn.gso-dim
  by fastforce

```

**lemma** *upper-tri:*

```

  fixes  $i :: \text{nat}$ 
  and  $j :: \text{nat}$ 
  assumes  $j > i$ 

```

```

assumes  $j < n$ 
shows  $((RAT\ M)!i) \cdot (Mt!j) = 0$ 
proof(-)
  have  $(gs.gso\ j) \cdot (RAT\ M)!i = 0$ 
    using gram-schmidt-fs-lin-indpt.gso-scalar-zero[of  $n\ (RAT\ M)\ j\ i$ ]
      Mt-gso-connect[of  $j$ ]
      assms
      M-locale-2
      more-dim
    by presburger
  then have  $(Mt!j) \cdot ((RAT\ M)!i) = 0$ 
    using Mt-gso-connect[of  $j$ ] assms
    by simp
  then show ?thesis
    using comm-scalar-prod[of  $(Mt!j)\ n\ ((RAT\ M)!i)$ ]
      carrier-vecI[of  $(Mt!j)\ n$ ]
      carrier-vecI[of  $((RAT\ M)!i)\ n$ ]
      access-index-M-dim[of  $i$ ]
      dim-vecs-in-Mt[of  $j$ ]
      assms
    by auto
qed
lemma one-diag:
  fixes  $i::nat$ 
  assumes  $0 \leq i$ 
  assumes  $i < n$ 
  shows  $((RAT\ M)!i) \cdot (Mt!i) = sq\ norm\ (Mt!i)$ 
proof(-)
  have  $mu: ((RAT\ M)!i) \cdot (Mt!i) = (gs.\mu\ i\ i) * sq\ norm\ (Mt!i)$ 
    using gram-schmidt-fs-lin-indpt.fi-scalar-prod-gso[of  $n\ (RAT\ M)\ i\ i$ ]
      M-locale-2
      assms
      more-dim
      Mt-gso-connect
    by presburger
  moreover have  $gs.\mu\ i\ i = 1$ 
    by (meson gs.\mu.elims order-less-imp-not-eq2)
  then show ?thesis
    using mu
    by fastforce
qed

```

```

lemma coord-invariance:
  fixes  $j::nat$ 
  fixes  $k::nat$ 
  fixes  $i::nat$ 
  assumes  $k \leq j$ 
  assumes  $j+i \leq n$ 

```

```

assumes  $k > 0$ 
shows  $(\text{lattice-coord } (s \ (j+i)))\$(n-k) = (\text{lattice-coord } (s \ j))\$(n-k)$ 
 $\wedge (s \ (j+i)) \cdot \text{Mt}!(n-k) = (s \ j) \cdot \text{Mt}!(n-k)$ 
using assms
proof(induct i)
  case 0
  show ?case by simp
next
  case (Suc i)
  have  $j+ \ (\text{Suc } i) = \text{Suc } (j+i)$  by simp
  then have  $1:s \ (\text{Suc } (j+i)) = s \ (j + (\text{Suc } i))$  by simp
  then have  $\text{sub}:s \ (\text{Suc } (j+i)) =$ 
 $(s \ (j+i)) - (\text{rat-of-int } (\text{calculate-c } (s \ (j+i)) \ \text{Mt} \ (\text{Suc } (j+i))))$ 
 $\cdot_v \ (\text{RAT } M)!(\text{dim-vec } (s \ (j+i)) - (\text{Suc } (j+i)))$ 
  using sub-s[of j+i] Suc(3) by linarith
  then have  $\text{dim1}: \text{dim-vec } (s \ (j + i)) = n$ 
  using s-dim[of j+i] using Suc(3) by auto
  then have  $\text{dim2}: \text{dim-vec}$ 
 $(\text{map of-int-hom.vec-hom } M !$ 
 $(\text{dim-vec } (s \ (j + i)) - \text{Suc } (j + i))) = n$ 
  using access-index-M-dim[of n - Suc (j + i)] Suc(3)
  by auto
  have  $k\text{-in-range}: 0 \leq (n-k) \wedge (n-k) < n$  using Suc(2) Suc(3) Suc(4)
  by simp
  have  $\text{index-in-range}: 0 \leq (\text{dim-vec } (s \ (j+i)) - (\text{Suc } (j+i))) \wedge (\text{dim-vec } (s \ (j+i))$ 
 $- (\text{Suc } (j+i))) < n$ 
  using Suc(3) s-dim[of j+i]
  by simp
  moreover have  $\text{carriers}: s \ (j+i) \in \text{carrier-vec } n \wedge$ 
 $\text{map of-int-hom.vec-hom } M ! (\text{dim-vec } (s \ (j + i)) - \text{Suc } (j +$ 
 $i)) \in \text{carrier-vec } n$ 
  using dim1 dim2
 $\text{carrier-vecI}[of s \ (j + i) \ n]$ 
 $\text{carrier-vecI}[of \text{map of-int-hom.vec-hom } M ! (\text{dim-vec } (s \ (j + i)) - \text{Suc } (j$ 
 $+ i)) \ n]$ 
  by fast

  let  $?s\text{Suc} = (s \ (\text{Suc } (j+i)))$ 
  let  $?si = (s \ (j+i))$ 
  let  $?c = (\text{rat-of-int } (\text{calculate-c } (s \ (j+i)) \ \text{Mt} \ (\text{Suc } (j+i))))$ 
  let  $?ind = (\text{dim-vec } (s \ (j+i)) - (\text{Suc } (j+i)))$ 

  have  $?si - ?c \cdot_v \ (\text{RAT } M)!\ ?ind = ?si + (-?c) \cdot_v \ (\text{RAT } M)!\ ?ind$ 
  using minus-add-uminus-vec[of ?si n ?c \cdot_v (RAT M)!\ ?ind]
 $\text{carriers}$ 
  by fastforce
  then have  $(\text{lattice-coord } (?si - ?c \cdot_v \ (\text{RAT } M)!\ ?ind))\$(n-k) =$ 
 $(\text{lattice-coord } (?si))\$(n-k) + (-?c) * (\text{lattice-coord } ((\text{RAT } M)!\ ?ind))\$(n-k)$ 
  using linear[of ?si (RAT M)!\ ?ind n-k - ?c] dim1 dim2 k-in-range

```

**by metis**  
**then have** *lin-lattice-coord*:(*lattice-coord* (?*sSuc*))\$(*n-k*) =  
(*lattice-coord*(?*si*))\$(*n-k*) - ?*c\** (*lattice-coord*((*RAT M*)!*?ind*))\$(*n-k*)  
**using sub**  
**by algebra**  
**have** *neq*:*Suc* (*j+i*) $\neq$ *k* **using** *Suc*( $\mathcal{Q}$ ) *Suc*( $\mathcal{Q}$ ) **by auto**  
**moreover have** ((*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))) $\neq$  (*n-k*)  
**using** *s-dim*[*of j+i*] *neq Suc*( $\mathcal{Q}$ )  
**by** (*metis Suc*( $\mathcal{Q}$ )  $\langle j + Suc\ i = Suc\ (j + i) \rangle$  *diff-0-eq-0 diff-cancel2*  
*diff-commute diff-diff-cancel diff-diff-eq diff-is-0-eq dim1*)  
**moreover have** (*lattice-coord* ((*RAT M*)!(*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))) )  
)\$(*n-k*)=  
(*unit-vec* *n* ( (*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))))\$(*n-k*)  
**using** *unit*[*of dim-vec* (*s* (*j+i*)) - (*Suc* (*j+i*))] *index-in-range* **by presburger**  
**then have** *zero*:(*lattice-coord* ((*RAT M*)!(*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))) )  
)\$(*n-k*) = 0  
**unfolding unit-vec-def**  
**using** *neq calculation*( $\mathcal{Q}$ ) *k-in-range* **by fastforce**  
**then have** (*lattice-coord* (*s* (*Suc* (*j+i*))))\$(*n-k*) = ( (*lattice-coord* (*s* (*j+i*)))\$(*n-k*))  
-  
(*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))) ) )  
\* 0  
**using zero lin-lattice-coord** **by presburger**  
**then have** *conclusion1*:(*lattice-coord* (*s* (*Suc* (*j+i*))) )\$(*n-k*) = ( (*lattice-coord*  
(*s* (*j+i*)))\$(*n-k*))  
**by simp**  
**have** *init-sub*:(*s* (*Suc* (*j+i*))) $\cdot$  *Mt*!(*n-k*) = ((*s* (*j+i*)) -  
((*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))) )  $\cdot_v$  (*RAT M*)!(*dim-vec* (*s*  
(*j+i*))) - (*Suc* (*j+i*))) ) ) )  
 $\cdot$  (*Mt*!(*n-k*))  
**using sub**  
**by simp**  
**moreover have** *carrier-prod*:( (*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))) ) )  
) )  
 $\cdot_v$  (*RAT M*)!(*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*)))  $\in$  *carrier-vec n*  
**using** *smult-carrier-vec*[*of* (*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))) ) )  
(*RAT M*)!(*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))) *n*] *carrier-vecI dim2* **by**  
*blast*  
**moreover have** *l*:((*s* (*j+i*)) -  
( (*rat-of-int* (*calculate-c* (*s* (*j+i*)) *Mt* (*Suc* (*j+i*))) ) )  $\cdot_v$  (*RAT M*)!(*dim-vec* (*s*  
(*j+i*))) - (*Suc* (*j+i*))) ) ) )  
 $\cdot$  (*Mt*!(*n-k*)) = (*s* (*j+i*)) $\cdot$  (*Mt*!(*n-k*)) - ( (*rat-of-int* (*calculate-c* (*s* (*j+i*))  
*Mt* (*Suc* (*j+i*))) ) ) )  
 $\cdot_v$  (*RAT M*)!(*dim-vec* (*s* (*j+i*))) - (*Suc* (*j+i*))) ) $\cdot$  (*Mt*!(*n-k*))  
**using** *s-dim*[*of j+i*]  
*assms*( $\mathcal{Q}$ )  
*access-index-M-dim*  
*dim-vecs-in-Mt*  
*carrier-vecI*[*of Mt*!(*n-k*) *n*]

$carrier-vecI[of (RAT M)!((dim-vec (s (j+i))) - (Suc (j+i))) n]$   
 $add-scalar-prod-distrib[of$   
 $(s (j+i))$   
 $n$   
 $(rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i))) ) \cdot_v (RAT M)! ( (dim-vec$   
 $(s (j+i))) - (Suc (j+i)) )$   
 $(Mt!(n-k))]$   
**using**  $calculation(5)$   $carriers k-in-range minus-scalar-prod-distrib$  **by**  $blast$

**moreover then have**  $lin-scalar-prod:((s (j+i)) -$   
 $( (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i))) ) \cdot_v (RAT M)! ( (dim-vec (s$   
 $(j+i))) - (Suc (j+i)) ) ) )$   
 $\cdot (Mt!(n-k)) = (s (j+i)) \cdot (Mt!(n-k)) - (rat-of-int (calculate-c (s (j+i)) Mt$   
 $(Suc (j+i)) ) )$   
 $* ((RAT M)! ( (dim-vec (s (j+i))) - (Suc (j+i))$   
 $) \cdot (Mt!(n-k)))$   
**by**  $(metis dim2 dim-vecs-in-Mt k-in-range scalar-prod-smult-left)$   
**moreover have**  $step-past-index:(dim-vec (s (j+i))) - (Suc (j+i)) < n-k$   
**using**  $s-dim[of j+i] Suc(3) Suc(2)$   
**by**  $(simp add: calculation(3) diff-le-mono2 dim1 le-SucI nat-less-le trans-le-add1)$   
**moreover have**  $( (RAT M)! ( (dim-vec (s (j+i))) - (Suc (j+i)) ) \cdot (Mt!(n-k)$   
 $) ) = 0$   
**using**  $step-past-index upper-tri[of (dim-vec (s (j+i))) - (Suc (j+i)) n-k] Suc(4)$   
**by**  $simp$   
**then have**  $(s (Suc (j+i))) \cdot Mt!(n-k) = (s (j+i)) \cdot Mt!(n-k) -$   
 $( (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i))) ) * 0)$   
**using**  $lin-scalar-prod init-sub$   
**by**  $algebra$   
**then have**  $conclusion2:(s (Suc (j+i))) \cdot Mt!(n-k) = (s (j+i)) \cdot Mt!(n-k)$  **by**  
 $auto$   
**show**  $?case$   
**by**  $(metis Suc(2) Suc(3) Suc(4) Suc.hyps Suc-leD \langle j + Suc i = Suc (j + i) \rangle$   
 $conclusion1 conclusion2)$   
**qed**

**lemma**  $small-orth-coord:$

**fixes**  $i::nat$   
**assumes**  $1 \leq i$   
**assumes**  $i \leq n$   
**shows**  $abs ( (s i) \cdot Mt!(n-i)) \leq (sq-norm (Mt!(n-i))) * (1/2)$   
**proof**(-)

**have**  $minus-plus:Suc (i-1) = i$  **using**  $assms(1)$  **by**  $auto$   
**then have**  $init-sub:s i = (s (i-1)) - ( (rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $\cdot_v (RAT M)! ( (dim-vec (s (i-1))) - i))$   
**using**  $sub-s[of i-1]$   
**by**  $(metis (full-types) Suc-le-eq assms(2) less-eq-nat.simps(1))$   
**then have**  $scalar-distrib:(s i) \cdot Mt!(n-i) = (s (i-1)) \cdot Mt!(n-i) - (( (rat-of-int$   
 $(calculate-c (s (i-1)) Mt i ) )$   
 $\cdot_v (RAT M)! ( (dim-vec (s (i-1))) - i)) \cdot Mt!(n-i))$

**using** *add-scalar-prod-distrib*[of  $(s (i-1)) n ( (rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $\cdot_v (RAT M)!( (dim-vec (s (i-1))) -i) Mt!(n-i)$   
 $s-dim$ [of  $i-1$ ]  
 $carrier-vecI$ [of  $Mt!(n-i)$ ]  
 $carrier-vecI$ [of  $(RAT M)!( (dim-vec (s (i-1))) -i)$ ]  
 $access-index-M-dim$ [of  $( (dim-vec (s (i-1))) -i)$ ]  
 $dim-vecs-in-Mt$ [of  $n-i$ ]  
 $init-sub$   
 $minus-scalar-prod-distrib$ [of  $(s (i-1)) n ( (rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $Mt i ) )$   
 $\cdot_v (RAT M)!( (dim-vec (s (i-1))) -i) Mt!(n-i)$   
**by** (*metis Suc-leD assms(2) diff-Suc-less gs.smult-closed le0 minus-plus non-trivial*)  
**also have** *scalar-commute*: $(s (i-1)) \cdot Mt!(n-i) - ((rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $(i-1)) Mt i ) )$   
 $\cdot_v (RAT M)!( (dim-vec (s (i-1)))$   
 $-i) \cdot Mt!(n-i) =$   
 $(s (i-1)) \cdot Mt!(n-i) - ( (rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $* (((RAT M)!( (dim-vec (s (i-1))) -i) \cdot Mt!(n-i) ) )$   
**using** *scalar-prod-smult-left*  
 $carrier-vecI$ [of  $Mt!(n-i)$ ]  
 $carrier-vecI$ [of  $(RAT M)!( (dim-vec (s (i-1))) -i)$ ]  
 $access-index-M-dim$   
 $dim-vecs-in-Mt$   
**by** (*smt (verit) Suc-le-D assms(2) diff-less index-minus-vec(2) index-smult-vec(2)*)  
 $init-sub minus-plus s-dim zero-less-Suc$   
**moreover have** *index-in-range*:  $0 \leq n-i \wedge n-i < n$   
**using** *assms(1) assms(2)*  
**by** *simp*  
**moreover have** *sq-norm-eq*: $((RAT M)!( (dim-vec (s (i-1))) -i) \cdot Mt!(n-i) =$   
 $sq-norm (Mt!(n-i))$   
**using** *one-diag*[of  $n-i$ ]  
 $s-dim$ [of  $i-1$ ]  
 $index-in-range$   
 $assms(1)$   
 $assms(2)$   
 $less-imp-diff-less$   
**by** *simp*  
**then have**  $(s i) \cdot Mt!(n-i) = (s (i-1)) \cdot Mt!(n-i) -$   
 $( (rat-of-int (calculate-c (s (i-1)) Mt i ) ) * sq-norm (Mt!(n-i)))$   
**using** *scalar-distrib scalar-commute sq-norm-eq* **by** *argo*  
**then have** *final-sub*: $abs((s i) \cdot Mt!(n-i)) = abs(( (rat-of-int (calculate-c (s (i-1)) Mt i ) )$   
 $(i-1)) Mt i ) )$   
 $* sq-norm (Mt!(n-i)) - (s (i-1)) \cdot$   
 $Mt!(n-i)$   
**using** *abs-minus-commute* **by** *simp*  
**then have** *round-small*: $abs(rat-of-int (calculate-c (s (i-1)) Mt i ) -$   
 $((s (i-1)) \cdot (Mt!( (dim-vec (s (i-1))) - i ) ) )$

$/ (sq\text{-norm-vec } (Mt!((dim\text{-vec } (s (i-1))) - i))) \leq 1/2$

by (metis calculate-c.simps of-int-round-abs-le)

moreover have pos:  $0 \leq sq\text{-norm } (Mt!(n-i))$

by (simp add: sq-norm-vec-ge-0)

then have  $(sq\text{-norm } (Mt!(n-i))) * abs((rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i) - ((s (i-1)) \cdot (Mt!((dim\text{-vec } (s (i-1))) - i))) / (sq\text{-norm-vec } (Mt!((dim\text{-vec } (s (i-1))) - i)))))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$

using pos round-small mult-left-mono by blast

then have 2:  $abs((sq\text{-norm } (Mt!(n-i))) * (rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i) - ((s (i-1)) \cdot (Mt!((dim\text{-vec } (s (i-1))) - i))) / (sq\text{-norm-vec } (Mt!((dim\text{-vec } (s (i-1))) - i)))))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$

using pos by (smt (verit) abs-mult abs-of-nonneg)

have  $i \leq n$

using assms(2) by simp

then have abs( $(sq\text{-norm } (Mt!(n-i))) * (rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i) - (sq\text{-norm } (Mt!(n-i))) * ((s (i-1)) \cdot (Mt!((dim\text{-vec } (s (i-1))) - i))) / (sq\text{-norm } (Mt!(n-i)))))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$ )

using 2

$s\text{-dim}[of i]$

by (smt (verit) Rings.ring-distrib(4) Suc-leD minus-plus s-dim)

then have 1:  $abs((sq\text{-norm } (Mt!(n-i))) * (rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i) - ((s (i-1)) \cdot (Mt!((dim\text{-vec } (s (i-1))) - i))) * ((sq\text{-norm } (Mt!(n-i))) / (sq\text{-norm } (Mt!(n-i)))))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$

using assms(2) s-dim

by (smt (z3) gs.cring-simprules(14) times-divide-eq-right)

moreover have nonzero:  $sq\text{-norm } (Mt!(n-i)) \neq 0$

using Mt-gso-connect[of n-i] assms

by (metis M-locale-2 gram-schmidt-fs-lin-indpt.sq-norm-pos index-in-range length-map rel-simps(70))

moreover have cancel:  $(sq\text{-norm } (Mt!(n-i))) / (sq\text{-norm } (Mt!(n-i))) = 1$

using nonzero

by auto

moreover have dim-match:  $dim\text{-vec } (s (i-1)) = n$

using s-dim[of i-1] assms(2)

by linarith

then have final-ineq:  $abs((sq\text{-norm } (Mt!(n-i))) * (rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i) - ((s (i-1)) \cdot (Mt!((dim\text{-vec } (s (i-1))) - i)))))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$

using 1 cancel

by (smt (verit) gs.r-one)

then have rearrange-final-ineq:  $abs((rat\text{-of-int } (calculate\text{-c } (s (i-1)) Mt i)))$

$$* (sq\text{-norm } (Mt!(n-i)) - ((s (i-1)) \cdot (Mt!(n-i)))) \leq (sq\text{-norm } (Mt!(n-i))) * (1/2)$$
**using** *dim-match*  
**by** *algebra*  
**show** *?thesis*  
**using** *final-sub rearrange-final-ineq*  
**by** *argo*  
**qed**  
**lemma** *lattice-carrier*:  $L \subseteq \text{carrier-vec } n$   
**proof** –  
**have**  $x \in \text{carrier-vec } n$  **if**  $x\text{-def}: x \in L$  **for**  $x$   
**proof** –  
**obtain**  $f$  **where**  $f\text{-def}: x = \text{sumlist } (\text{map } (\lambda i. (f i) \cdot_v M!i) [0..<n])$   
**using**  $x\text{-def}$  **unfolding**  $L\text{-def}$  *lattice-of-def* **by** *fast*  
**have**  $(f i) \cdot_v M!i \in \text{carrier-vec } n$  **if**  $0 \leq i \wedge i < n$  **for**  $i$   
**using** *access-index-M-dim*[*of i*]  
**by** (*metis carrier-vec-dim-vec map-carrier-vec nth-map smult-closed that*)  
**then have**  $\text{set } (\text{map } (\lambda i. (f i) \cdot_v M!i) [0..<n]) \subseteq \text{carrier-vec } n$  **by** *auto*  
**then have**  $\text{sumlist } (\text{map } (\lambda i. (f i) \cdot_v M!i) [0..<n]) \in \text{carrier-vec } n$  **by** *simp*  
**then show**  $x \in \text{carrier-vec } n$  **using**  $f\text{-def}$  **by** *fast*  
**qed**  
**then show** *?thesis* **by** *fast*  
**qed**

## 4 Lattice Lemmas

**lemma** *lattice-sum-close*:  
**fixes**  $u::\text{int vec}$  **and**  $v::\text{int vec}$   
**assumes**  $u \in L$   $v \in L$   
**shows**  $u + v \in L$   
**proof** –  
**let**  $?mM = \text{mat-of-cols } n$   $M$   
**have**  $1: ?mM \in \text{carrier-mat } n$  **using** *dim-vecs-in-M* **by** *fastforce*  
**have**  $\text{set-}M: \text{set } M \subseteq \text{carrier-vec } n$   
**using** *dim-vecs-in-M carrier-vecI* **by** *blast*  
**have**  $\text{as-mat-mult:lattice-of } M = \{y \in \text{carrier-vec } n. \exists x \in \text{carrier-vec } n. ?mM *_v x = y\}$   
**using** *lattice-of-as-mat-mult*[*OF set-M*] **by** *blast*  
**then obtain**  $u1$  **where**  $u1\text{-def}: u = ?mM *_v u1 \wedge u1 \in \text{carrier-vec } n$  **using** *assms* **unfolding**  $L\text{-def}$  **by** *auto*  
**obtain**  $v1$  **where**  $v1\text{-def}: v = ?mM *_v v1 \wedge v1 \in \text{carrier-vec } n$   
**using** *assms as-mat-mult* **unfolding**  $L\text{-def}$  **by** *auto*  
**have**  $u1 + v1 \in \text{carrier-vec } n$  **using**  $u1\text{-def}$   $v1\text{-def}$  **by** *blast*  
**moreover have**  $?mM *_v (u1 + v1) = u + v$   
**using**  $u1\text{-def}$   $v1\text{-def}$  *1 mult-add-distrib-mat-vec*[*of ?mM n n u1 v1*]  
**by** *metis*  
**moreover have**  $u + v \in \text{carrier-vec } n$  **using** *assms lattice-carrier* **by** *blast*  
**ultimately show**  $u + v \in L$   
**using** *as-mat-mult* **unfolding**  $L\text{-def}$

by *blast*  
qed

**lemma** *lattice-smult-close*:  
fixes  $u::\text{int vec}$  and  $q::\text{int}$   
assumes  $u \in L$   
shows  $q \cdot_v u \in L$

**proof** –  
let  $?mM = \text{mat-of-cols } n \ M$   
have  $1: ?mM \in \text{carrier-mat } n \ n$  using *dim-vecs-in-M* by *fastforce*  
have  $\text{set-}M: \text{set } M \subseteq \text{carrier-vec } n$   
using *dim-vecs-in-M carrier-vecI* by *blast*  
have  $\text{as-mat-mult:lattice-of } M = \{y \in \text{carrier-vec } n. \exists x \in \text{carrier-vec } n. ?mM * _v x = y\}$   
using *lattice-of-as-mat-mult[OF set-M]* by *blast*  
then obtain  $v::\text{int vec}$  where  $v\text{-def}: u = ?mM * _v v \wedge v \in \text{carrier-vec } n$   
using *assms unfolding L-def* by *auto*  
then have  $q \cdot_v v \in \text{carrier-vec } n$  by *blast*  
moreover then have  $q \cdot_v u = ?mM * _v (q \cdot_v v)$  using  $1 \ v\text{-def}$  by *fastforce*  
ultimately show  $q \cdot_v u \in L$   
by (*metis (mono-tags, lifting) L-def as-mat-mult assms mem-Collect-eq smult-closed*)  
qed

**lemma** *smult-vec-zero*:  
fixes  $v :: 'a::\text{ring vec}$   
shows  $0 \cdot_v v = 0_v (\text{dim-vec } v)$   
unfolding *smult-vec-def vec-eq-iff*  
by (*auto*)

**lemma** *coset-s*:  
fixes  $i::\text{nat}$   
assumes  $i \leq n$   
shows  $s \ i \in \text{coset}$   
using *assms*  
**proof** (*induct i*)  
case  $0$   
have  $s \ 0 = -t$  unfolding *s-def* by *simp*  
moreover have  $\text{carrier-}mt: -t \in \text{carrier-vec } n$  using *length-M carrier-vecI* [of  $t \ n$ ] by *fastforce*  
ultimately have  $pzero: s \ 0 = \text{of-int-hom.vec-hom } (0_v \ n) - t$  by *fastforce*  
let  $?zero = \lambda j. 0$   
have  $0 < \text{length } M$  using *non-trivial* by *fast*  
then have  $M!0 \in \text{set } M$  by *force*  
then have  $M!0 \in L$  using *basis-in-latticeI* [of  $M \ M!0$ ] *dim-vecs-in-M carrier-vecI*  
*L-def*  
by *blast*  
then have  $0_v \ n \in L$

```

using lattice-smult-close[of M!0 0] smult-vec-zero[of M!0] access-index-M-dim[of
0] non-trivial
unfolding L-def
by fastforce
then show ?case using pzero by blast
next
case (Suc i)
let ?q = (rat-of-int (calculate-c (s i) Mt (Suc i) ) )
let ?ind = ( (dim-vec (s i)) - (Suc i))
have sub:s (Suc i) = (s i) - (?q ·v (RAT M)!?ind)
using sub-s[of i] Suc.prem by linarith
have s i ∈ coset using Suc by auto
then obtain x where x-def:x ∈ L ∧ (s i) = of-int-hom.vec-hom x - t by blast
have (?q ·v (RAT M)!?ind) ∈ of-int-hom.vec-hom' L
proof-
have dim-vec (s i) = n using s-dim[of i] Suc.prem by fastforce
then have in-range: ?ind < n ∧ 0 ≤ ?ind using Suc.prem by simp
then have com-hom: (RAT M)!(?ind) = of-int-hom.vec-hom (M! ?ind) by auto
have M! ?ind ∈ set M using in-range by simp
then have mil:M! ?ind ∈ L using basis-in-latticeI[of M M! ?ind] dim-vecs-in-M
carrier-vecI L-def
by blast
moreover have ?q ·v (of-int-hom.vec-hom (M! ?ind)) =
of-int-hom.vec-hom ((calculate-c (s i) Mt (Suc i) ) ·v M! ?ind)
by fastforce
moreover have (calculate-c (s i) Mt (Suc i) ) ·v M! ?ind ∈ L
using lattice-smult-close[of M! ?ind (calculate-c (s i) Mt (Suc i) )] mil by
simp
ultimately show ( ?q ·v (RAT M)!?ind) ∈ of-int-hom.vec-hom' L
using com-hom
by force
qed
then obtain y where y-def: ( ?q ·v (RAT M)!?ind) = of-int-hom.vec-hom y ∧
y ∈ L by blast
have carrier-x: x ∈ carrier-vec n using lattice-carrier x-def by blast
have carrier-y: y ∈ carrier-vec n using lattice-carrier y-def by blast
then have carrier-my: -y ∈ carrier-vec n by simp
then have 1:-( ?q ·v (RAT M)!?ind) = of-int-hom.vec-hom (-y) using y-def
by fastforce
then have s (Suc i) = of-int-hom.vec-hom x - t + of-int-hom.vec-hom (-y)
using sub x-def y-def 1 by fastforce
then have s (Suc i) = of-int-hom.vec-hom x + of-int-hom.vec-hom (-y) - t
using lattice-carrier x-def y-def length-M
by fastforce
moreover have of-int-hom.vec-hom x + of-int-hom.vec-hom (-y) = of-int-hom.vec-hom
(x + -y)
using carrier-my carrier-x by fastforce
ultimately have 2: s (Suc i) = of-int-hom.vec-hom (x + -y) - t
by metis

```

**have**  $-y = -1 \cdot_v y$  **by** *auto*  
**then have**  $-y \in L$  **using** *lattice-smult-close y-def* **by** *simp*  
**then have**  $x + -y \in L$  **using** *lattice-sum-close x-def* **by** *simp*  
**then show** *?case* **using** 2 **by** *fast*  
**qed**

**lemma** *subtract-coset-into-lattice*:

**fixes**  $v::\text{rat } \text{vec}$   
**fixes**  $w::\text{rat } \text{vec}$   
**assumes**  $v \in \text{coset}$   
**assumes**  $w \in \text{coset}$   
**shows**  $v - w \in \text{of-int-hom.vec-hom}'L$   
**proof** –  
**obtain**  $l1$  **where**  $l1\text{-def}: v = l1 - t \wedge l1 \in \text{of-int-hom.vec-hom}'L$  **using** *assms(1)*  
**by** *blast*  
**obtain**  $l2$  **where**  $l2\text{-def}: w = l2 - t \wedge l2 \in \text{of-int-hom.vec-hom}'L$  **using** *assms(2)*  
**by** *blast*  
**have**  $\text{carrier-l1}: l1 \in \text{carrier-vec } n$  **using** *lattice-carrier l1-def* **by** *force*  
**have**  $\text{carrier-l2}: l2 \in \text{carrier-vec } n$  **using** *lattice-carrier l2-def* **by** *force*  
**obtain**  $l1p$  **where**  $l1p\text{-def}: l1 = \text{of-int-hom.vec-hom } l1p \wedge l1p \in L$  **using**  $l1\text{-def}$   
**by** *fast*  
**obtain**  $l2p$  **where**  $l2p\text{-def}: l2 = \text{of-int-hom.vec-hom } l2p \wedge l2p \in L$  **using**  $l2\text{-def}$   
**by** *fast*  
**have**  $-l2p = -1 \cdot_v l2p$  **using**  $\text{carrier-l2}$  **by** *fastforce*  
**then have**  $ml2p:-l2p \in L$  **using** *lattice-smult-close[of l2p -1]*  $l2p\text{-def}$  **by** *presburger*  
**then have**  $\text{of-int-hom.vec-hom } (-l2p) \in \text{of-int-hom.vec-hom}'L$  **by** *simp*  
**moreover have**  $\text{of-int-hom.vec-hom } (-l2p) = -l2$  **using**  $l2p\text{-def}$  **by** *fastforce*  
**then have**  $l1 - l2 = \text{of-int-hom.vec-hom } (l1p - l2p)$  **using**  $l1p\text{-def } l2p\text{-def}$   
*carrier-l1 carrier-l2* **by** *auto*  
**moreover have**  $l1p - l2p \in L$  **using** *lattice-sum-close[of l1p -l2p]*  
 $l1p\text{-def } l2p\text{-def } ml2p$  *carrier-l1 carrier-l2*  
**by** (*simp add: minus-add-uminus-vec*)  
**ultimately have**  $l1 - l2 \in \text{of-int-hom.vec-hom}'L$  **by** *fast*  
**moreover have**  $v - w = l1 - l2$  **using**  $l1\text{-def } l2\text{-def}$  *length-M carrier-vecI*  
*carrier-l1 carrier-l2* **by** *force*  
**ultimately show** *?thesis* **by** *simp*  
**qed**

**lemma** *t-in-coset*:

**shows**  $\text{uminus } t \in \text{coset}$   
**using** *coset-s[of 0]* *babai-help.simps* **unfolding**  $s\text{-def}$  **by** *simp*

## 5 Lemmas on closest distance

**lemma** *closest-distance-sq-pos*:  $\text{closest-distance-sq} \geq 0$

**proof** –

**have**  $\forall N \in \{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\}. 0 \leq N$   
**using** *sq-norm-vec-ge-0* **by** *auto*  
**moreover have**  $\{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\} \neq \{\}$  **using** *t-in-coset*

**by** *blast*  
**ultimately have**  $0 \leq \text{Inf } \{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\}$   
**by** (*meson cInf-greatest*)  
**then show** *?thesis unfolding closest-distance-sq-def by blast*  
**qed**

**definition** *witness:: rat vec  $\Rightarrow$  rat  $\Rightarrow$  bool*  
**where** *witness*  $v \ \varepsilon\text{-closest} = (\text{sq-norm } v \leq \varepsilon\text{-closest} \wedge v \in \text{coset} \wedge \text{dim-vec } v = n)$

**definition** *close-condition::rat  $\Rightarrow$  bool*  
**where** *close-condition*  $\varepsilon\text{-closest} \equiv$   
*(if*  $\text{closest-distance-sq} = 0$  *then*  $0 \leq \text{real-of-rat } \varepsilon\text{-closest}$   
*else*  $\text{real-of-rat } \varepsilon\text{-closest} > \text{closest-distance-sq}$ )  
 $\wedge (\text{real-of-rat } \varepsilon\text{-closest} \leq \varepsilon * \text{closest-distance-sq})$

**lemma** *close-rat:*  
**obtains**  $\varepsilon\text{-closest}::\text{rat}$   
**where** *close-condition*  $\varepsilon\text{-closest}$   
**proof**(*cases closest-distance-sq = 0*)  
**case** *t:True*  
**then have**  $\varepsilon * \text{closest-distance-sq} = \text{real-of-rat } (0::\text{rat})$  **by** *simp*  
**then have**  $\text{real-of-rat } (0::\text{rat}) \leq \varepsilon * \text{closest-distance-sq} \wedge \text{closest-distance-sq} \leq (\text{real-of-rat } (0::\text{rat}))$   
**using** *t by force*  
**then show** *?thesis*  
**using** *that t unfolding close-condition-def bymetis*  
**next**  
**case** *f:False*  
**then have**  $0 < \text{closest-distance-sq}$   
**using** *closest-distance-sq-pos by linarith*  
**moreover have**  $(1::\text{real}) < \varepsilon$  **using**  $\varepsilon$  **by** *simp*  
**ultimately have**  $\text{closest-distance-sq} < \varepsilon * \text{closest-distance-sq}$  **by** *simp*  
**then show** *?thesis*  
**using** *Rats-dense-in-real[of closest-distance-sq  $\varepsilon * \text{closest-distance-sq}$ ] that unfolding close-condition-def by (metis Rats-cases less-eq-real-def)*  
**qed**

**definition**  $\varepsilon\text{-closest}::\text{rat}$   
**where**  $\varepsilon\text{-closest} = (\text{if } \exists r. \text{close-condition } r \text{ then } \text{SOME } r. \text{close-condition } r \text{ else } 0)$

**lemma**  $\varepsilon\text{-closest-lemma: close-condition } \varepsilon\text{-closest}$   
**using** *close-rat unfolding  $\varepsilon\text{-closest-def by (metis (full-types))$*

**lemma** *rational-tri-ineq:*  
**fixes**  $v::\text{rat vec}$   
**fixes**  $w::\text{rat vec}$

**assumes**  $\dim\text{-vec } v = \dim\text{-vec } w$   
**shows**  $(\text{sq-norm } (v+w)) \leq 4 * (\text{Max } \{(\text{sq-norm } v), (\text{sq-norm } w)\})$   
**proof** –  
**let**  $?d = \dim\text{-vec } w$   
**let**  $?M = \text{Max } \{(\text{sq-norm } v), (\text{sq-norm } w)\}$   
**have**  $\text{carr-v}:v \in \text{carrier-vec } ?d$  **using**  $\text{assms carrier-vecI}[of v ?d]$  **by**  $\text{fastforce}$   
**have**  $\text{carr-w}:w \in \text{carrier-vec } ?d$  **using**  $\text{carrier-vecI}[of w ?d]$  **by**  $\text{fastforce}$   
**have**  $\text{carr-vw}:v+w \in \text{carrier-vec } ?d$  **using**  $\text{carr-v carr-w add-carrier-vec}$  **by**  $\text{blast}$   
**have**  $\text{sq-norm } (v+w) = (v+w) \cdot (v+w)$   
**by**  $(\text{simp add: sq-norm-vec-as-cscalar-prod})$   
**also have**  $(v+w) \cdot (v+w) = v \cdot (v+w) + w \cdot (v+w)$   
**using**  $\text{add-scalar-prod-distrib}[of v ?d w v+w]$   
 $\text{carr-v carr-w carr-vw}$  **by**  $\text{blast}$   
**also have**  $v \cdot (v+w) + w \cdot (v+w) = v \cdot v + v \cdot w + w \cdot v + w \cdot w$   
**using**  $\text{scalar-prod-add-distrib}[of v ?d v w]$   
 $\text{scalar-prod-add-distrib}[of w ?d v w]$   
 $\text{carr-v carr-w carr-vw}$  **by**  $\text{algebra}$   
**also have**  $v \cdot w = w \cdot v$   
**using**  $\text{carr-v carr-w comm-scalar-prod}$  **by**  $\text{blast}$   
**also have**  $v \cdot v = \text{sq-norm } v$   
**using**  $\text{sq-norm-vec-as-cscalar-prod}[of v]$  **by**  $\text{force}$   
**also have**  $w \cdot w = \text{sq-norm } w$   
**using**  $\text{sq-norm-vec-as-cscalar-prod}[of w]$  **by**  $\text{force}$   
**finally have**  $\text{sq-norm } (v+w) = \text{sq-norm } v + \text{sq-norm } w + 2 * (w \cdot v)$  **by**  $\text{force}$   
**also have**  $b1: \text{sq-norm } v \leq ?M$  **by**  $\text{force}$   
**also have**  $b2: \text{sq-norm } w \leq ?M$  **by**  $\text{force}$   
**also have**  $2 * (w \cdot v) \leq 2 * (\text{Max } \{(\text{sq-norm } v), (\text{sq-norm } w)\})$   
**proof** –  
**have**  $(w \cdot v) \wedge 2 \leq (\text{sq-norm } v) * (\text{sq-norm } w)$   
**using**  $\text{scalar-prod-Cauchy}[of w ?d v]$   $\text{carr-w carr-v}$  **by**  $\text{algebra}$   
**also have**  $(\text{sq-norm } v) * (\text{sq-norm } w) \leq ?M * ?M$   
**using**  $b1 b2 \text{sq-norm-vec-ge-0}[of w] \text{sq-norm-vec-ge-0}[of v]$   
 $\text{mult-mono}[of \text{sq-norm } v ?M \text{sq-norm } w ?M]$  **by**  $\text{linarith}$   
**also have**  $?M * ?M = ?M \wedge 2$   
**using**  $\text{power2-eq-square}[of ?M]$  **by**  $\text{presburger}$   
**finally have**  $(w \cdot v) \wedge 2 \leq ?M \wedge 2$  **by**  $\text{blast}$   
**also have**  $(w \cdot v) \wedge 2 = \text{abs}(w \cdot v) \wedge 2$  **by**  $\text{force}$   
**finally have**  $\text{abs}(w \cdot v) \wedge 2 \leq ?M \wedge 2$  **by**  $\text{presburger}$   
**moreover have**  $0 \leq \text{abs}(w \cdot v)$  **by**  $\text{fastforce}$   
**moreover have**  $0 \leq ?M$   
**using**  $\text{sq-norm-vec-ge-0}[of w] \text{sq-norm-vec-ge-0}[of v]$  **by**  $\text{fastforce}$   
**ultimately have**  $\text{abs}(w \cdot v) \leq ?M$   
**using**  $\text{power2-le-imp-le}$  **by**  $\text{blast}$   
**also have**  $(w \cdot v) \leq \text{abs}(w \cdot v)$  **by**  $\text{force}$   
**finally show**  $?thesis$  **by**  $\text{linarith}$   
**qed**  
**finally show**  $?thesis$  **by**  $\text{auto}$   
**qed**

```

lemma witness-exists:
  shows  $\exists v. \text{witness } v \ \varepsilon\text{-closest}$ 
proof(cases closest-distance-sq = 0)
  case t: True
  have  $\varepsilon\text{-closest} = 0$ 
  using  $\varepsilon\text{-closest-lemma } t$ 
  unfolding witness-def unfolding close-condition-def
  by auto
  then have equiv: ?thesis =  $(\exists v. v \in \text{coset} \wedge (\text{dim-vec } v = n) \wedge (\text{sq-norm } v) \leq 0)$ 
  unfolding witness-def  $\varepsilon\text{-closest-def}$  by auto
  show ?thesis
  proof(rule ccontr)
    assume contra:  $\neg ?thesis$ 
    have  $\{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\} \neq \{\}$  using t-in-coset by fast
    then have limit-point:  $\exists v::\text{rat vec. real-of-rat } (\text{sq-norm } v) < (\text{eps}::\text{real}) \wedge v \in \text{coset}$ 
  if  $0 < \text{eps}$  for eps
    using t cInf-lessD[of  $\{\text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset}\} \ \text{eps}$ ] that
    unfolding closest-distance-sq-def by auto
    moreover have  $0 < \text{real-of-rat } ((\text{sq-norm } ((\text{RAT } M)!0)) / (4 * \alpha^{n-1}))$ 
  proof-
    have  $0 < 1 / (4 * \alpha^{n-1})$  using non-trivial alpha by force
    moreover have  $0 < (\text{sq-norm } ((\text{RAT } M)!0))$ 
      using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M 0]
      gram-schmidt-fs-lin-indpt.sq-norm-gso-le-f[of n RAT M 0]
      M-locale-2 non-trivial
    by fastforce
    ultimately show ?thesis by auto
  qed
  ultimately obtain  $v::\text{rat vec}$  where v-def:  $\text{real-of-rat } (\text{sq-norm } v) < \text{real-of-rat } ((\text{sq-norm } ((\text{RAT } M)!0)) / (4 * \alpha^{n-1})) \wedge v \in \text{coset}$ 
  by presburger
  then have  $\text{dim-vec } v = n$ 
  using length-M by force
  then have  $0 < \text{real-of-rat } (\text{sq-norm } v)$ 
  using equiv contra v-def by auto
  then obtain  $w::\text{rat vec}$  where w-def:  $\text{real-of-rat } (\text{sq-norm } w) < \text{real-of-rat } (\text{sq-norm } v) \wedge w \in \text{coset}$ 
  using limit-point by fast
  then have small-w:  $\text{real-of-rat } (\text{sq-norm } w) < \text{real-of-rat } ((\text{sq-norm } ((\text{RAT } M)!0)) / (4 * \alpha^{n-1}))$ 
  using v-def by argo
  have lat:  $w - v \in \text{of-int-hom.vec-hom } L$  using subtract-coset-into-lattice[of w v]
  using v-def w-def by force
  then obtain l where l-def:  $l \in L \wedge w - v = \text{of-int-hom.vec-hom } l$  by blast
  then have  $\text{of-int-hom.vec-hom } l \in \text{gs.lattice-of } (\text{RAT } M)$ 
  using lattice-of-of-int[of M n l] dim-vecs-in-M carrier-vecI L-def by blast
  then have lat-hom:  $w - v \in \text{gs.lattice-of } (\text{RAT } M)$  using l-def by simp
  have  $\text{sq-norm } v \neq \text{sq-norm } w$  using w-def by auto

```

**then have**  $neq:w \neq v$  **by** *meson*  
**have**  $c1:w \in \text{carrier-vec } n$  **using** *length-M w-def lattice-carrier carrier-dim-vec*  
**by** *fastforce*  
**moreover have**  $c2:v \in \text{carrier-vec } n$  **using** *length-M v-def lattice-carrier carrier-dim-vec* **by** *fastforce*  
**ultimately have**  $c3:w-v \in \text{carrier-vec } n$  **by** *simp*  
**have**  $neqzero:w-v \neq 0_v$   $n$   
**proof**(*rule ccontr*)  
**assume**  $c:\neg ?thesis$   
**have**  $w-v=0_v$   $n$  **using**  $c$  **by** *blast*  
**then have**  $w=v+0_v$   $n$  **using**  $c1$   $c2$   $c3$   
**by** (*smt (verit, ccfv-SIG) gs.M.add.r-inv-ex minus-add-minus-vec minus-cancel-vec minus-zero-vec right-zero-vec*)  
**then show** *False* **using**  $c2$   $neq$  **by** *simp*  
**qed**  
**then have**  $w-v \in \text{gs.lattice-of } (RAT\ M) - \{0_v\}$   $n$  **using** *lat-hom* **by** *blast*  
**moreover have**  $\alpha^{n-1} * (\text{sq-norm } (w-v)) < (\text{sq-norm } ((RAT\ M)!0))$   
**proof**-  
**have**  $w-v = w+(-v)$  **by** *fastforce*  
**then have**  $\text{sq-norm } (w-v) = \text{sq-norm } (w+(-v))$  **by** *simp*  
**also have**  $\text{sq-norm } (w+(-v)) \leq 4 * \text{Max}\{\text{sq-norm } w, \text{sq-norm } (-v)\}$   
**using** *rational-tri-ineq[of w -v]*  $c1$   $c2$  **by** *fastforce*  
**also have**  $\text{sq-norm } (-v) = \text{sq-norm } v$   
**proof**-  
**have**  $-v = (-1) \cdot_v v$  **by** *fastforce*  
**then have**  $\text{sq-norm } (-v) = ((-1) \cdot_v v) \cdot ((-1) \cdot_v v)$  **using** *sq-norm-vec-as-cscalar-prod[of -v]* **by** *force*  
**then have**  $\text{sq-norm } (-v) = (-1) * (-1) * (v \cdot v)$  **using**  $c1$   $c2$  **by** *simp*  
**then show** *?thesis* **using** *sq-norm-vec-as-cscalar-prod[of v]* **by** *simp*  
**qed**  
**also have**  $\text{Max}\{\text{sq-norm } w, \text{sq-norm } (v)\} < ((\text{sq-norm } ((RAT\ M)!0)) / (4 * \alpha^{n-1}))$   
**using** *v-def small-w of-rat-less* **by** *auto*  
**finally have**  $\text{sq-norm } (w-v) < 4 * ((\text{sq-norm } ((RAT\ M)!0)) / (4 * \alpha^{n-1}))$   
**by** *linarith*  
**then have**  $\text{sq-norm } (w-v) < (\text{sq-norm } ((RAT\ M)!0)) / (\alpha^{n-1})$  **by** *linarith*  
**moreover have**  $p:0 < \alpha^{n-1}$  **using** *alpha* **by** *fastforce*  
**ultimately show** *?thesis* **using**  $p$   
**by** (*metis gs.cring-simprules(14) pos-less-divide-eq*)  
**qed**  
**ultimately show** *False*  
**using** *gram-schmidt-fs-lin-indpt.weakly-reduced-imp-short-vector[of n (RAT M)  $\alpha$  w-v]*  
*M-locale-2 reduced alpha*  
**unfolding** *gs.reduced-def L-def* **by** *force*  
**qed**  
**next**  
**case** *False*  
**then have** *closest-distance-sq < real-of-rat  $\varepsilon$ -closest*

**using**  $\varepsilon$ -closest-lemma **unfolding**  $\varepsilon$ -closest-def close-condition-def  
**by** presburger  
**moreover have**  $\{ \text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset} \} \neq \{ \}$  **using**  $t$ -in-coset  
**by** fast  
**ultimately obtain**  $l$  **where**  $l \in \{ \text{real-of-rat } (\text{sq-norm } x::\text{rat}) \mid x. x \in \text{coset} \} \wedge l < \text{real-of-rat } \varepsilon$ -closest  
**using** closest-distance-sq-pos  
**unfolding** closest-distance-sq-def  
**by** (meson cInf-lessD)  
**moreover then obtain**  $v::\text{rat vec}$  **where**  $l = \text{real-of-rat } (\text{sq-norm } v) \wedge v \in \text{coset}$   
**by** blast  
**ultimately show** ?thesis **unfolding** witness-def lattice-carrier  
**by** (smt (verit) length-M index-minus-vec(2) mem-Collect-eq of-rat-less-eq)  
**qed**

## 6 More linear algebra lemmas

**lemma** carrier-Ms:

**shows**  $\text{mat-}M \in \text{carrier-mat } n \ n$   $\text{mat-}M\text{-inv} \in \text{carrier-mat } n \ n$   
**using**  $M$ -dim  $M$ -inv-dim  
**apply** blast  
**by** (simp add:  $M$ -inv-dim(1)  $M$ -inv-dim(2) carrier-matI)

**lemma** carrier-L:

**fixes**  $v::\text{rat vec}$   
**assumes**  $\text{dim-vec } v = n$   
**shows**  $\text{lattice-coord } v \in \text{carrier-vec } n$   
**unfolding** lattice-coord-def  
**using**  $\text{mult-mat-vec-carrier}$ [of  $\text{mat-}M\text{-inv } n \ n \ v$ ]  
 $\text{carrier-Ms}$   
 $\text{carrier-vecI}$ [of  $v$ ]  
 $\text{assms}(1)$   
**by** fast

**lemma** sumlist-index-commute:

**fixes**  $Lst::\text{rat vec list}$   
**fixes**  $i::\text{nat}$   
**assumes**  $\text{set } Lst \subseteq \text{carrier-vec } n$   
**assumes**  $i < n$   
**shows**  $(\text{gs.sumlist } Lst)\$i = \text{sum-list } (\text{map } (\lambda j. (Lst!j)\$i) [0..<(\text{length } Lst)])$   
**using**  $\text{assms}$   
**proof**(induct  $Lst$ )  
**case** Nil  
**have**  $\text{gs.sumlist } Nil = 0_v \ n$  **using**  $\text{assms}$  **unfolding**  $\text{gs.sumlist-def}$  **by** auto  
**then have**  $\text{lhs}:(\text{gs.sumlist } Nil)\$i = 0$  **using**  $\text{assms}(2)$  **by** auto  
**have**  $[0..<(\text{length } Nil)] = Nil$  **by** simp  
**then have**  $(\text{map } (\lambda j. (Nil!j)\$i) [0..<(\text{length } Nil)]) = Nil$  **by** blast  
**then have**  $\text{sum-list } (\text{map } (\lambda j. (Nil!j)\$i) [0..<(\text{length } Nil)]) = 0$  **by** simp  
**then show** ?case **using** lhs **by** simp

```

next
  case (Cons a Lst)
  let ?CaLst = Cons a Lst
  have set Lst  $\subseteq$  carrier-vec n using Cons.prem by auto
  then have carr:gs.sumlist Lst  $\in$  carrier-vec n using assms gs.sumlist-carrier[of
Lst ]
    by blast
  have gs.sumlist (Cons a Lst) = a + gs.sumlist Lst by simp
  then have lhs:(gs.sumlist ?CaLst)$i = a$i + (gs.sumlist Lst)$i using assms
carr by simp
  have sum-list (map ( $\lambda j$ . (?CaLst!j)$i) [0..\lambda l. l$i) ?CaLst)
    by (smt (verit) length-map map-eq-conv' map-nth nth-map)
  moreover have sum-list (map ( $\lambda l$ . l$i) ?CaLst) = a$i + sum-list (map ( $\lambda l$ . l$i)
Lst) by simp
  moreover have sum-list (map ( $\lambda l$ . l$i) Lst) = sum-list (map ( $\lambda j$ . (Lst!j)$i)
[0..\lambda j. (Lst!j)$i) [0..

```

lemma mat-mul-to-sum-list:

```

fixes A::rat mat
fixes v::rat vec
assumes dim-vec v = dim-col A
assumes dim-row A = n
shows  $A*_v v = gs.sumlist (map ( $\lambda j$ . v$j  $\cdot_v$  (col A j)) [0 ..< dim-col A])$ 
```

proof –

```

  have carrier:set (map ( $\lambda j$ . v $ j  $\cdot_v$  col A j) [0..\subseteq Rn
  by (smt (verit) assms(2) carrier-dim-vec dim-col ex-map-conv index-smult-vec(2)
subset-code(1))
  have (A*_v v)$i = gs.sumlist (map ( $\lambda j$ . v$j  $\cdot_v$  (col A j)) [0 ..< dim-col A])$i if
small:i<dim-row A for i
  proof –
    let ?rAi = row A i

    have 1:(A*_v v)$i = ?rAi  $\cdot$  v using small by simp
    have 2:?rAi  $\cdot$  v = sum-list (map ( $\lambda j$ . (?rAi$j)*(v$j)) [0..\cdot_v (col A j))$i if jsmall:j<dim-col A for j
      unfolding row-def col-def using small jsmall
    by force
    then have (map ( $\lambda j$ . (?rAi$j)*(v$j)) [0..\lambda j. (v$j  $\cdot_v$  (col
A j))$i) [0..

```



using pre by presburger  
qed

lemma *sumlist-linear-coord*:

fixes *Lst*::int vec list  
 assumes  $\bigwedge i. i < \text{length } Lst \implies \text{dim-vec } (Lst!i) = n$   
 shows *lattice-coord* (map-vec rat-of-int (sumlist *Lst*)) = *gs.sumlist* (map *lattice-coord* (RAT *Lst*))  
 using *assms*  
 proof(induct *Lst*)  
 case Nil  
 have *rhs*:*gs.sumlist*(map *lattice-coord* (RAT Nil)) =  $0_v \ n$  by *fastforce*  
 have *map-vec rat-of-int* (sumlist Nil) =  $0_v \ n$  by *auto*  
 then have *lattice-coord* (map-vec rat-of-int (sumlist Nil)) =  $0_v \ n$   
 unfolding *lattice-coord-def* using *M-inv-dim*  
 by (metis *carrier-Ms*(2) *gs.M.add.r-cancel-one'* *gs.M.zero-closed mult-add-distrib-mat-vec mult-mat-vec-carrier*)  
 then show ?*case* using *rhs* by *simp*  
 next  
 case (Cons *a Lst*)  
 let ?*CaLst* = Cons *a Lst*  
 let ?*ra* = *of-int-hom.vec-hom a*  
 have *dim*: $i \in \text{set } ?CaLst \implies \text{dim-vec } i = n$  for *i* using *Cons.prem*s  
 by (metis *in-set-conv-nth*)  
 then have *i-lt*: ( $i < \text{length } Lst \implies \text{dim-vec } (Lst ! i) = n$ ) for *i*  
 using *Cons.prem*s *carrier-dim-vec* by *auto*  
 have *carrier*: $\text{set } ?CaLst \subseteq \text{carrier-vec } n$  using *Cons.prem*s  
 using *carrier-vecI dim* by *fast*  
 then have *carrier-sumCaLst*: (sumlist ?*CaLst*) $\in \text{carrier-vec } n$  by *force*  
 have *carrier-a*:  $a \in \text{carrier-vec } n$  using *carrier* by *force*  
 have *carrier-Lst*: $\text{set } Lst \subseteq \text{carrier-vec } n$  using *carrier* by *simp*  
 have *lhs*:*lattice-coord* (map-vec rat-of-int (sumlist ?*CaLst*)) = (*lattice-coord* ?*ra*)  
 + *gs.sumlist* (map *lattice-coord* (RAT *Lst*))  
 proof-  
 have *carrier-sumLst*: *sumlist Lst* $\in \text{carrier-vec } n$  using *carrier-Lst* by *force*  
 have *sumlist ?CaLst* = *a* + *sumlist Lst* by *force*  
 then have (map-vec rat-of-int (sumlist ?*CaLst*)) = ?*ra* + (map-vec rat-of-int (sumlist *Lst*))  
 using *carrier-a carrier-sumLst carrier-sumCaLst* by *auto*  
 then have *lattice-coord* (map-vec rat-of-int (sumlist ?*CaLst*))  
 = *lattice-coord*(?*ra*) + *lattice-coord*(map-vec rat-of-int (sumlist *Lst*))  
 unfolding *lattice-coord-def*  
 using *carrier-sumCaLst carrier-a carrier-sumLst*  
 by (metis *carrier-Ms*(2) *map-carrier-vec mult-add-distrib-mat-vec*)  
 then show ?*thesis* using *i-lt Cons.hyps*  
 by *algebra*  
 qed  
 moreover have *rhs*:*gs.sumlist* (map *lattice-coord* (RAT ?*CaLst*)) =  
 (*lattice-coord* ?*ra*) + *gs.sumlist* (map *lattice-coord* (RAT *Lst*))

by *fastforce*  
ultimately show *?case* by *argo*  
qed

lemma *integral-sum*:

fixes *l::nat*  
assumes  $\bigwedge j1. j1 < l \implies$   
 $map\ f\ [0..<l]!\ j1 \in \mathbb{Z}$   
shows *sum-list*  
 $(map\ f\ [0..<l]) \in \mathbb{Z}$   
using *assms*  
proof(*induct l*)  
case 0  
have  $(map\ f\ [0..<0]) = Nil$  by *auto*  
then have *sum-list*  $(map\ f\ [0..<0]) = 0$  by *simp*  
then show *?case* by *simp*  
next  
case (*Suc l*)  
have *nontriv*:*Suc l* > 0 by *simp*  
have *break*:*sum-list*  $(map\ f\ [0..<(Suc\ l)]) = sum-list\ (map\ f\ [0..<l]) + (f\ l)$  by  
*fastforce*  
have *l* < *Suc l* by *simp*  
then have  $[0..<(Suc\ l)]!l = l$   
by (*metis nth-upt plus-nat.add-0*)  
moreover then have  $f\ ([0..<(Suc\ l)]!\ l) = (map\ f\ [0..<(Suc\ l)]!\ l)$   
by (*metis One-nat-def Suc-diff-Suc diff-Suc-1 local.nontriv nat-SN.default-gt-zero*)  
 $nth-map-upt\ nth-upt\ plus-1-eq-Suc\ real-add-less-cancel-right-pos$   
ultimately have *z*:*f l*  $\in \mathbb{Z}$  using *Suc.prem*s by *fastforce*  
have  $\bigwedge j1. j1 < l \implies$   
 $map\ f\ [0..<l]!\ j1 \in \mathbb{Z}$   
by (*metis Suc.prem*s *diff-Suc-1'* *diff-Suc-Suc less-SucI nth-map-upt*)  
then have *sum-list*  $(map\ f\ [0..<l]) \in \mathbb{Z}$  using *Suc* by *blast*  
then show *?case* using *z break* by *force*  
qed

lemma *int-coord*:

fixes *i::nat*  
assumes  $0 \leq i$   
assumes  $i < n$   
fixes *v::int vec*  
assumes  $v \in L$   
assumes *dim-vec v = n*  
shows (*lattice-coord*  $(map-vec\ rat-of-int\ v)$ ) $\$i \in \mathbb{Z}$   
proof –  
obtain *w* where *w-def*:*v = sumlist*  $(map\ (\lambda i. of-int\ (w\ i) \cdot_v\ M!\ i)\ [0..<length\ M])$

```

    using L-def assms(β) vec-module.lattice-of-def
  by blast
let ?Lst = (map (λ i. of-int (w i) ·v M ! i) [0 ..< length M])
have dims-j:dim-vec (?Lst!j) = n if j-lt:j<length ?Lst for j
  using access-index-M-dim carrier-vecI j-lt by force
let ?recover = (map lattice-coord (RAT ?Lst))
have 1:lattice-coord (map-vec rat-of-int v) = gs.sumlist ?recover
  using sumlist-linear-coord[of ?Lst]
    w-def
    dims-j
  by blast
have int-recover:∧j. j<n⇒(?recover!j)$i ∈ℤ∧ (dim-vec (?recover!j)) = n
proof -
  fix j::nat
  assume small:j<n
  have ?recover!j = lattice-coord ((RAT ?Lst)!j)
    using List.nth-map[of j (RAT ?Lst) lattice-coord]
      small
  by simp
  then have ?recover!j = lattice-coord (of-int-hom.vec-hom (?Lst!j))
    using List.nth-map[of j ?Lst of-int-hom.vec-hom]
      small
  by simp
  then have ?recover!j = lattice-coord (of-int-hom.vec-hom (of-int (w j) ·v M !
j))
    using List.nth-map[of j [0 ..< length M] (λ i. of-int (w i) ·v M ! i)]
      small
  by simp
  then have commuted-maps:?recover!j = mat-M-inv *v (of-int-hom.vec-hom
(of-int (w j) ·v M ! j))
    unfolding lattice-coord-def
  by simp
  then have ?recover!j = mat-M-inv *v((of-int (of-int (w j))) ·v of-int-hom.vec-hom
(M ! j))
    using of-int-hom.vec-hom-smult[of of-int (w j) M ! j]
  by metis
  then have ?recover!j = (of-int (of-int (w j))) ·v (mat-M-inv *v of-int-hom.vec-hom
(M ! j))
    using mult-mat-vec[of mat-M-inv n n of-int-hom.vec-hom (M ! j) (of-int
(of-int (w j)))]
      carrier-Ms
      access-index-M-dim[of j]
      carrier-vecI[of of-int-hom.vec-hom (M ! j) n]
  by (simp add: small)
  then have ?recover!j = (of-int (of-int (w j))) ·v (lattice-coord (of-int-hom.vec-hom
(M ! j)))
    unfolding lattice-coord-def
  by simp
  then have recover-unit:?recover!j = (of-int (of-int (w j))) ·v (unit-vec n j)

```

```

    using unit[of j]
      small
    by simp
  then have (?recover!)$i=((of-int (of-int (w j))) ·v (unit-vec n j))$i
    by simp
  then have (?recover!)$i = (of-int (of-int (w j))) * (unit-vec n j)$i
    by (simp add: assms(2))
  then have (?recover!)$i = (of-int (of-int (w j))) * (if i=j then 1 else 0)
    using small assms(2)
    by simp
  moreover have (if i=j then 1 else 0) ∈ ℤ
    by simp
  moreover have (of-int (of-int (w j))) ∈ ℤ
    by simp
  moreover have dim-vec (?recover!) = n
  using recover-unit
    smult-closed[of (unit-vec n j) (of-int (of-int (w j)))]
    unit-vec-carrier[of n j]
  by force
  ultimately show (?recover!)$i ∈ ℤ ∧ dim-vec (?recover!) = n
    by simp
qed
then have ∀ v ∈ set ?recover. dim-vec v = n
  by auto
then have set ?recover ⊆ carrier-vec n
  using carrier-vecI
  by blast
then have (gs.sumlist ?recover)$i = sum-list (map (λj. (?recover!)$i) [0..

```

```

qed
ultimately have (gs.sumlist ?recover)$i∈ℤ
  using integral-sum[of n (λj. map lattice-coord
    (map of-int-hom.vec-hom (map (λi. of-int (w i) ·v M ! i) [0..<n])) !
      j $
      i)]
  by argo
then show ?thesis
  using 1
  by simp
qed

```

lemma *int-coord-for-rat*:

```

fixes i::nat
assumes 0 ≤ i
assumes i < n
fixes v::rat vec
assumes v ∈ of-int-hom.vec-hom' L
assumes dim-vec v = n
shows (lattice-coord v)$i ∈ ℤ
proof -
  let ?hom = of-int-hom.vec-hom
  obtain vint where v = ?hom vint ∧ vint ∈ L using assms(3) by blast
  moreover then have (lattice-coord (?hom vint))$i ∈ ℤ using int-coord assms by
  simp
  ultimately show ?thesis by simp
qed

```

## 7 Coord-Invariance

This section shows that the algorithm output matches true closest (or near-closest) vector in some trailing coordinates.

definition *I* where

$$I = (\text{if } (\{i \in \{0..<n\}. ((sq\text{-}norm (M!i)::rat)) \leq 4 * \epsilon\text{-closest}\}::nat\ set) \neq \{\} \\ \text{then } Max (\{i \in \{0..<n\}. ((sq\text{-}norm (M!i)::rat)) \leq 4 * \epsilon\text{-closest}\}::nat\ set) \text{ else } -1)$$

lemma *I-geq*:

```

shows I ≥ -1
unfolding I-def
by simp

```

lemma *I-leq*:

```

shows I < n
unfolding I-def
by force

```

lemma *index-geq-I-big*:

```

fixes  $i::nat$ 
assumes  $i > I$ 
assumes  $i < n$ 
shows  $((sq\text{-norm } (Mt!i)::rat)) > 4 * \varepsilon\text{-closest}$ 
proof(rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $((sq\text{-norm } (Mt!i)::rat)) \leq 4 * \varepsilon\text{-closest}$  by linarith
  then have  $i\text{-def}: i \in (\{i \in \{0..<n\}. ((sq\text{-norm } (Mt!i)::rat)) \leq 4 * \varepsilon\text{-closest}\}::nat\ set)$ 
using assms by fastforce
  then have  $(\{i \in \{0..<n\}. ((sq\text{-norm } (Mt!i)::rat)) \leq 4 * \varepsilon\text{-closest}\}::nat\ set) \neq \{\}$  by
fast
  moreover then have  $I = Max (\{i \in \{0..<n\}. ((sq\text{-norm } (Mt!i)::rat)) \leq 4 * \varepsilon\text{-closest}\}::nat\ set)$ 
unfolding I-def by presburger
  moreover have finite  $(\{i \in \{0..<n\}. ((sq\text{-norm } (Mt!i)::rat)) \leq 4 * \varepsilon\text{-closest}\}::nat\ set)$ 
by simp
  ultimately show False using assms i-def eq-Max-iff by auto
qed

```

**lemma** *scalar-prod-gs-from-lattice-coord*:

```

fixes  $i::nat$ 
fixes  $v::rat\ vec$ 
assumes  $dim\text{-vec } v = n$ 
assumes  $i < n$ 
shows  $v \cdot Mt!i = sum\text{-list } (map (\lambda k. (lattice\text{-coord } v)\$k * (((RAT\ M)!k) \cdot Mt!i))$ 
 $[i..<n])$ 
proof( $-$ )
  let  $?lc = lattice\text{-coord } v$ 
  let  $?recover = ((map (\lambda j. ?lc\$j \cdot_v (RAT\ M)!j) [0 ..<n]))$ 
  let  $?gsv = Mt!i$ 
  have  $v = gs.sumlist\ ?recover$ 
    using recover-from-lattice-coord[of v] assms
    by blast
  then have split-ip:  $v \cdot ?gsv = (gs.sumlist (map (\lambda j. ?lc\$j \cdot_v (RAT\ M)!j) [0 ..<n])) \cdot ?gsv$ 
    by simp
  have  $\bigwedge u. u \in set\ ?recover \implies u \in carrier\text{-vec } n$ 
proof( $-$ )
    fix  $u::rat\ vec$ 
    assume  $u\text{-init}: u \in set\ ?recover$ 
    then have index-small:  $find\text{-index } ?recover\ u < length\ ?recover$ 
      by (meson find-index-leq-length)
    then have carrier-v-ind-M:  $(RAT\ M)!(find\text{-index } ?recover\ u) \in carrier\text{-vec } n$ 
      using carrier-vecI[of (RAT M)!(find-index ?recover u) n]
      access-index-M-dim
    by (smt (z3) M-locale-1 gram-schmidt-fs-Rn.f-carrier length-map map-nth)
    then have  $u = ?recover!(find\text{-index } ?recover\ u)$ 
      using u-init
    by (simp add: find-index-in-set)

```

```

then have  $u = (\lambda j. ?lc\$j \cdot_v (RAT\ M)!j) (find-index\ ?recover\ u)$ 
using u-init
      List.nth-map[of find-index ?recover u [0..<n] (\lambda j. ?lc\$j \cdot_v (RAT\ M)!j)]
      index-small
by auto
then have  $u = ?lc\$(find-index\ ?recover\ u) \cdot_v (RAT\ M)!(find-index\ ?recover\ u)$ 
by simp
then show  $u \in carrier\text{-}vec\ n$ 
using carrier-v-ind-M
      smult-carrier-vec[of ?lc\$(find-index\ ?recover\ u) (RAT\ M)!(find-index
?recover\ u) n]
by presburger
qed
then have  $result\text{-}sumlist\text{-}L : v \cdot ?gsv = sum\text{-}list (map (\lambda w. w \cdot ?gsv) ?recover)$ 
using split-ip
      gs.scalar-prod-left-sum-distrib[of ?recover ?gsv]
by (metis (no-types, lifting) assms(2) carrier-dim-vec dim-vecs-in-Mt)
let  $?L = (map (\lambda w. w \cdot ?gsv) ?recover)$ 
have  $2 : \bigwedge k. k < n \implies ?L!k = ?lc\$k * ((RAT\ M)!k \cdot ?gsv)$ 
proof(-)
  fix  $k :: nat$ 
  assume  $k\text{-}bound : k < n$ 
  then have  $?L!k = (\lambda w. w \cdot ?gsv) (?recover!k)$ 
  by force
  then have  $?L!k = ?recover!k \cdot ?gsv$ 
  by simp
  then have  $?L!k = ((\lambda j. (?lc\$j \cdot_v (RAT\ M)!j)) k) \cdot ?gsv$ 
  using List.nth-map[of k [0..<n] (\lambda j. (?lc\$j \cdot_v (RAT\ M)!j))] k-bound
  by simp
  then have  $?L!k = (?lc\$k \cdot_v (RAT\ M)!k) \cdot ?gsv$ 
  by simp
  then show  $?L!k = ?lc\$k * ((RAT\ M)!k \cdot ?gsv)$ 
  using smult-scalar-prod-distrib[of (RAT\ M)!k n ?gsv ?L!k]
      access-index-M-dim
      dim-vecs-in-Mt[of i]
      carrier-vecI[of ?gsv n]
      k-bound
      assms
  by force
qed
moreover have  $length\ ?L = n$ 
by fastforce
ultimately have  $1 : ?L = (map (\lambda k. ?lc\$k * ((RAT\ M)!k \cdot ?gsv)) [0..<n])$ 
by auto
moreover then have  $filt : \bigwedge k. k < i \implies (\lambda k. ?lc\$k * ((RAT\ M)!k \cdot ?gsv)) k = 0$ 
proof(-)
  fix  $k :: nat$ 
  assume  $tri : k < i$ 
  then have  $(?gsv \cdot (RAT\ M)!k) = 0$ 

```

```

using gram-schmidt-fs-lin-indpt.gso-scalar-zero[of n (RAT M) i k]
      M-locale-2
      Mt-gso-connect[of i]
      assms(2)
      more-dim
by presburger
then have ((RAT M)!k)•?gsv = 0
using comm-scalar-prod[of ((RAT M)!k) n ?gsv ]
      access-index-M-dim[of k]
      tri
      assms(2)
      dim-vecs-in-Mt[of i]
      carrier-vecI[of ?gsv] carrier-vecI[of ((RAT M)!k)]
by fastforce
then have ?lc$k * ((RAT M)!k• ?gsv) = 0
by simp
then show (λk. ?lc$k * ((RAT M)!k• ?gsv)) k = 0
by blast
qed
ultimately have sum-list ?L = sum-list (map (λk. ?lc$k * ((RAT M)!k• ?gsv))
(filter (λk. i≤k) [0..using sum-list-map-filter[of [0..by (metis (no-types, lifting) le-eq-less-or-eq nat-neq-iff)
moreover have (filter (λk. i≤k) [0..using assms(2) bot-nat-0.extremum filter-upt
by presburger
ultimately have sum-list ?L = sum-list (map (λk. ?lc$k * ((RAT M)!k• ?gsv))
[i..by presburger
then show ?thesis
using result-sumlist-L
by simp
qed

```

```

lemma correct-coord-help:
  fixes i::nat
  assumes i < (int n) - I
  assumes witness v (ε-closest)
  assumes 0 < i
  shows (lattice-coord (s i))$(n-i) = (lattice-coord v)$(n-i)
      ∧ ( (s i) • Mt!(n-i) = v • Mt!(n-i) )
  using assms
proof(induct i rule: less-induct)
case (less i)
let ?lcs = (lattice-coord (s i))
let ?lcIs = λi. lattice-coord (s i)$(n-i)
let ?lcv = lattice-coord v
let ?gsv = Mt!(n-(i))

```

```

have leq:(int n)→I≤n+1
  using I-geq
  by simp
moreover have nonbase:0<i
  using less by blast
then have 1:i≤n
  using leq less
  by linarith
moreover have nms:n-(i)<n
  using 1 nonbase by linarith
ultimately have s-ip:(s (i)) · ?gsv = sum-list (map (λj. ?lcs$j *((RAT M)!j·
?gsv)) [n-(i)..<n])
  using scalar-prod-gs-from-lattice-coord[of s (i) n-(i)]
    s-dim[of i] by force
have dim-v:dim-vec v = n
  using assms(2)
  unfolding witness-def
  by blast
then have v-ip:v · ?gsv = sum-list (map (λj. ?lcv$j *((RAT M)!j· ?gsv))
[n-(i)..<n])
  unfolding witness-def
  using scalar-prod-gs-from-lattice-coord[of v n-i]
    nms assms(2)
    carrier-vecI[of v n]
  by satx
have [n-i..<n]≠[] using nms by auto
then have split-indices: [n-(i)..<n] = (n-i) # [n-(i)+1..<n]
  by (simp add: upt-eq-Cons-conv)
then have split-s-list:(map (λj. ?lcs$j *((RAT M)!j· ?gsv)) [n-(i)..<n]) =
  ((λj. ?lcs$j *((RAT M)!j· ?gsv)) (n-(i)))#(map (λj. ?lcs$j *((RAT M)!j·
?gsv)) [n-(i)+1..<n])
  by simp
then have split-s-ip-pre:(s (i)) · ?gsv = ((λj. ?lcs$j *((RAT M)!j· ?gsv))
(n-(i)))
  + sum-list (map (λj. ?lcs$j *((RAT M)!j·
?gsv)) [n-(i)+1..<n])
  using s-ip
  by force
then have split-s-ip: (s (i)) · ?gsv = ((λj. ?lcs$j *((RAT M)!j· ?gsv)) (n-(i)))
  + sum-list (map (λj. ?lcs$j *((RAT M)!j·
?gsv)) [n-i+1..<n])
  by presburger
have split-v-list:(map (λj. ?lcv$j *((RAT M)!j· ?gsv)) [n-(i)..<n]) =
  ((λj. ?lcv$j *((RAT M)!j· ?gsv)) (n-(i)))#(map (λj. ?lcv$j *((RAT M)!j·
?gsv)) [n-(i)+1..<n])
  using split-indices by simp
then have split-v-ip-pre:v · ?gsv = ((λj. ?lcv$j *((RAT M)!j· ?gsv)) (n-(i)))
  + sum-list (map (λj. ?lcv$j *((RAT M)!j· ?gsv)) [n-(i)+1..<n])
  using v-ip

```



**then have**  $(s\ i) \cdot ?gsv - v \cdot ?gsv = ((?lcs\$(n-i) - ?lcv\$(n-i)) * ((RAT\ M)!(n-i) \cdot ?gsv))$   
**by algebra**  
**then have**  $case-2-from-case-1:(s\ i) \cdot ?gsv - v \cdot ?gsv = ((?lcs\$(n-i) - ?lcv\$(n-i)) * (sq-norm\ ?gsv))$   
**using one-diag[of n-i] 1 nms**  
**by fastforce**  
**then have**  $abs\ ((s\ i) \cdot ?gsv - v \cdot ?gsv) = abs(?lcs\$(n-i) - ?lcv\$(n-i)) * abs(sq-norm\ ?gsv)$   
**using abs-mult by auto**  
**then have**  $a:abs\ ((s\ i) \cdot ?gsv - v \cdot ?gsv) = abs(?lcs\$(n-i) - ?lcv\$(n-i)) * (sq-norm\ ?gsv)$   
**by (metis abs-of-nonneg sq-norm-vec-ge-0)**  
**have**  $lattice-coord-equal:?lcs\$(n-i) - ?lcv\$(n-i) = 0$   
**proof(rule ccontr)**  
**assume**  $\neg(?lcs\$(n-i) - ?lcv\$(n-i) = 0)$   
**then have**  $contra:?lcs\$(n-i) - ?lcv\$(n-i) \neq 0$  **by simp**  
**have**  $?lcs\$(n-i) - ?lcv\$(n-i) = (?lcs - ?lcv)\$(n-i)$   
**using index-minus-vec(1)[of n-i ?lcv ?lcs]**  
 $dim-preserve-lattice-coord[of v]$   
 $assms(2) nms$   
**unfolding witness-def by argo**  
**moreover have**  $?lcs - ?lcv = lattice-coord((s\ i) - v)$   
**using mult-minus-distrib-mat-vec**  
**unfolding lattice-coord-def**  
**by (metis 1 carrier-Ms(2) carrier-vecI dim-v s-dim)**  
**ultimately have**  $use-linear:?lcs\$(n-i) - ?lcv\$(n-i) = (lattice-coord((s\ i) - v))\$(n-i)$   
**by presburger**  
**have**  $(s\ i) - v \in of-int-hom.vec-hom'\ L$   
**using subtract-coset-into-lattice[of s i v]**  
 $coset-s[of i]$   
 $1\ assms(2)$   
**unfolding witness-def**  
**by linarith**  
**then have**  $use-int-coord:(lattice-coord((s\ i) - v))\$(n-i) \in \mathbb{Z}$   
**using int-coord-for-rat[of n-i ((s i) - v)] 1 nms**  
**by (simp add: dim-v)**  
**then have**  $abs((lattice-coord((s\ i) - v))\$(n-i)) > 0$   
**using contra use-linear**  
**by linarith**  
**then have**  $abs((lattice-coord((s\ i) - v))\$(n-i)) \geq 1$   
**using use-int-coord**  
**by (simp add: Ints-nonzero-abs-ge1 contra use-linear)**  
**then have**  $abs(?lcs\$(n-i) - ?lcv\$(n-i)) \geq 1$   
**using use-linear by presburger**  
**then have**  $abs(?lcs\$(n-i) - ?lcv\$(n-i)) * (sq-norm\ ?gsv) \geq sq-norm\ ?gsv$   
**using sq-norm-vec-ge-0[of ?gsv] mult-left-mono[of 1 abs(?lcs\\$(n-i) - ?lcv\\$(n-i)) sq-norm\ ?gsv] by algebra**  
**then have**  $big1:abs\ ((s\ i) \cdot ?gsv - v \cdot ?gsv) \geq sq-norm\ ?gsv$

**using** *a* **by** *argo*  
**then have** *tri-ineq*:  $\text{abs}(v \cdot ?gsv) \geq \text{abs}(\text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv))$   
**using** *cancel-ab-semigroup-add-class.diff-right-commute*  
*cancel-comm-monoid-add-class.diff-cancel diff-zero* **by** *linarith*  
**then have** *smallhalf*:  $\text{abs}((s \ i) \cdot ?gsv) \leq (1/2) * (\text{sq-norm } ?gsv)$   
**using** *small-orth-coord[of i] nonbase 1*  
**by** *fastforce*  
**then have**  $\text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv) \geq \text{sq-norm } ?gsv - (1/2) * (\text{sq-norm } ?gsv)$   
**using** *big1* **by** *linarith*  
**then have** *big2*:  $\text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv) \geq (1/2) * (\text{sq-norm } ?gsv)$   
**by** *linarith*  
**then have**  $\text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv) \geq 0$   
**using** *sq-norm-vec-ge-0[of ?gsv]* **by** *linarith*  
**then have**  $\text{abs}(\text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv)) = \text{abs}((s \ i) \cdot ?gsv - v \cdot ?gsv) - \text{abs}((s \ i) \cdot ?gsv)$   
**by** *fastforce*  
**then have**  $\text{abs}(v \cdot ?gsv) \geq (1/2) * (\text{sq-norm } ?gsv)$   
**using** *big2*  
**by** *linarith*  
**moreover have**  $(1/2) * (\text{sq-norm } ?gsv) \geq 0$   
**using** *sq-norm-vec-ge-0[of ?gsv]* **by** *simp*  
**moreover have**  $\text{abs}(v \cdot ?gsv) \geq 0$  **by** *simp*  
**ultimately have**  $\text{abs}(v \cdot ?gsv)^{\wedge} 2 \geq ((1/2) * (\text{sq-norm } ?gsv))^{\wedge} 2$   
**using** *nonneg-power-le* **by** *blast*  
**moreover have**  $(\text{sq-norm } v) * (\text{sq-norm } ?gsv) \geq \text{abs}(v \cdot ?gsv)^{\wedge} 2$   
**using** *scalar-prod-Cauchy[of v n ?gsv]*  
*carrier-vecI[of v n] assms(2)*  
*carrier-vecI[of ?gsv] dim-vecs-in-Mt[of n-i] nms*  
**unfolding** *witness-def*  
**by** *fastforce*  
**ultimately have**  $\text{sq-norm } v * \text{sq-norm } ?gsv \geq ((1/2) * (\text{sq-norm } ?gsv))^{\wedge} 2$   
**by** *order*  
**then have**  $\text{sq-norm } v * \text{sq-norm } ?gsv \geq (1/2)^{\wedge} 2 * (\text{sq-norm } ?gsv)^{\wedge} 2$   
**by** *(metis gs.nat-pow-distrib)*  
**then have**  $\text{sq-norm } v * \text{sq-norm } ?gsv \geq 1/4 * (\text{sq-norm } ?gsv)^{\wedge} 2$   
**by** *(smt (z3) numeral-Bit0-eq-double one-power2 power2-eq-square times-divide-times-eq)*  
**moreover have**  $\text{sq-norm } ?gsv > 0$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M n-i]*  
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]*  
*nms* **by** *force*  
**ultimately have** *big*:  $\text{sq-norm } v \geq 1/4 * \text{sq-norm } ?gsv$   
**by** *(simp add: power2-eq-square)*  
**have**  $n - i > I$   
**using** *less* **by** *linarith*  
**then have** *big-again*:  $\text{sq-norm } ?gsv > 4 * \varepsilon$  *closest*  
**using** *index-geq-I-big[of n-i] nms* **by** *simp*

```

then have sq-norm v > 1/4 * 4 * ε-closest
  using big by fastforce
then have sq-norm v > ε-closest by auto
then show False
  using assms(2)
  unfolding witness-def
  by linarith
qed
then have piece1: lattice-coord (s i) $ (n - i) = lattice-coord v $ (n - i)
  using lattice-coord-equal by simp
have (s i) · ?gsv - v · ?gsv = 0
  using lattice-coord-equal case-2-from-case-1
  by algebra
then show ?case using piece1 by simp
qed

```

```

lemma correct-coord:
  fixes v::rat vec
  fixes k::nat
  assumes witness v ε-closest
  assumes I < k
  assumes k < n
  shows (s n) · Mt!(k) = v · Mt!(k)
proof -
  have (s n) · Mt!(k) = (s (n-k)) · Mt!(k)
    using coord-invariance[of n-k n-k k] assms
    by force
  moreover have (s (n-k)) · Mt!(k) = v · Mt!(k)
    using correct-coord-help[of n-k v] assms
    by simp
  ultimately show ?thesis by simp
qed

```

## 8 Bound on distance to target, with $\epsilon$ factor.

This section culminates in a bound (which includes the  $\epsilon$  factor) on the output's distance to the target vector.

```

lemma sq-norm-from-Mt:
  fixes v::rat vec
  assumes v-carr:v∈carrier-vec n
  shows sq-norm v = sum-list (map (λi. (v·Mt!i) ^ 2 / (sq-norm (Mt!i))) [0..<n])
proof -
  let ?Mt-inv-list = map (λi. (1/sq-norm(Mt!i))·v (Mt!i)) [0..<n]
  have nonsing: ?Mt-inv-list!i ∈ carrier-vec n if i:0 ≤ i ∧ i < n for i
  proof -
    have 0 < sq-norm(Mt!i)
      using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]
      M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] i

```

by (simp add: M-locale-2)  
 then have  $0 < 1/\text{sq-norm}(Mt!i)$  by fastforce  
 then have  $(1/\text{sq-norm}(Mt!i)) \cdot_v (Mt!i) \in \text{carrier-vec } n$   
 using carrier-vecI[of (Mt!i)] dim-vecs-in-Mt[of i] i by blast  
 moreover have  $?Mt\text{-inv-list}!i = (1/\text{sq-norm}(Mt!i)) \cdot_v (Mt!i)$   
 using i by simp  
 ultimately show ?thesis by argo  
 qed  
 let  $?Mt\text{-inv-mat} = \text{mat-of-rows } n \ ?Mt\text{-inv-list}$   
 have carrier-mat-inv:  $?Mt\text{-inv-mat} \in \text{carrier-mat } n \ n$  by fastforce  
 let  $?vMt = ?Mt\text{-inv-mat} *_v v$   
 have  $?vMt\$i = ((1/\text{sq-norm}(Mt!i)) \cdot_v (Mt!i)) \cdot_v$  if  $i:0 \leq i \wedge i < n$  for i  
 using i nonsing[of i] by auto  
 have  $\text{dim-vMt}:\text{dim-vec } ?vMt = n$   
 using carrier-mat-inv v-carr by auto  
 let  $?Mt\text{-mat} = \text{mat-of-cols } n \ Mt$   
 have  $l:\text{length } Mt = n$   
 using gs.gram-schmidt-result[of RAT M Mt] basis dim-vecs-in-M  
 unfolding gs.lin-indpt-list-def  
 by fastforce  
 then have carrier-mat-Mt:  $?Mt\text{-mat} \in \text{carrier-mat } n \ n$   
 using dim-vecs-in-Mt carrier-vecI by auto  
 then have to-sumlist:  $?Mt\text{-mat} *_v ?vMt = \text{gs.sumlist } (\text{map } (\lambda j. ?vMt\$j \cdot_v (\text{col } ?Mt\text{-mat } j)) [0 ..< n])$   
 using mat-mul-to-sum-list[of ?vMt ?Mt-mat] dim-vMt  
 by fastforce  
 have  $?vMt\$i \cdot_v (\text{col } ?Mt\text{-mat } i) = (1/\text{sq-norm}(Mt!i)) * ((Mt!i) \cdot_v) \cdot_v Mt!i$  if  $i:0 \leq i \wedge i < n$  for i  
 using i l dim-vecs-in-Mt v-carr carrier-vecI by fastforce  
 then have  $(\text{map } (\lambda j. ?vMt\$j \cdot_v (\text{col } ?Mt\text{-mat } j)) [0 ..< n])$   
 $= (\text{map } (\lambda j. (1/\text{sq-norm}(Mt!j)) * ((Mt!j) \cdot_v) \cdot_v Mt!j) [0 ..< n])$   
 by simp  
 then have  $1:\text{gs.sumlist } (\text{map } (\lambda j. ?vMt\$j \cdot_v (\text{col } ?Mt\text{-mat } j)) [0 ..< n])$   
 $= \text{gs.sumlist } (\text{map } (\lambda j. (1/\text{sq-norm}(Mt!j)) * ((Mt!j) \cdot_v) \cdot_v Mt!j) [0 ..< n])$   
 by presburger  
 then have  $2:?Mt\text{-mat} *_v ?vMt = \text{gs.sumlist } (\text{map } (\lambda j. (1/\text{sq-norm}(Mt!j)) * ((Mt!j) \cdot_v) \cdot_v Mt!j) [0 ..< n])$   
 using to-sumlist by argo  
 have  $?Mt\text{-mat} *_v ?vMt = (?Mt\text{-mat} * ?Mt\text{-inv-mat}) *_v v$   
 using carrier-mat-Mt carrier-mat-inv v-carr by auto  
 have  $(?Mt\text{-inv-mat} * ?Mt\text{-mat})\$\$(i,j) = (1_m \ n)\$\$(i,j)$   
 if sensible-indices:  $0 \leq i \wedge i < n \wedge 0 \leq j \wedge j < n$  for i j  
 proof –  
 have  $(?Mt\text{-inv-mat} * ?Mt\text{-mat})\$\$(i,j) = (\text{row } ?Mt\text{-inv-mat } i) \cdot (\text{col } ?Mt\text{-mat } j)$   
 using sensible-indices carrier-mat-Mt carrier-mat-inv by auto  
 then have  $(?Mt\text{-inv-mat} * ?Mt\text{-mat})\$\$(i,j) = ?Mt\text{-inv-list}!i \cdot Mt!j$   
 using sensible-indices carrier-mat-Mt carrier-mat-inv nonsing  
 by auto

**then have**  $(?Mt\text{-inv-mat} * ?Mt\text{-mat})\$(i,j) = ((1/sq\text{-norm}(Mt!i)) \cdot_v (Mt!i)) \cdot Mt!j$   
**using** *sensible-indices* **by** *simp*  
**then have**  $(?Mt\text{-inv-mat} * ?Mt\text{-mat})\$(i,j) = (1/sq\text{-norm}(Mt!i)) * ((Mt!i) \cdot (Mt!j))$   
**using** *dim-vecs-in-Mt[of i]* *dim-vecs-in-Mt[of j]* *sensible-indices* **by** *auto*  
**moreover have**  $(1/sq\text{-norm}(Mt!i)) * ((Mt!i) \cdot (Mt!j)) = (\text{if } i=j \text{ then } 1 \text{ else } 0)$   
**proof** (*cases i=j*)  
**case** *diag: True*  
**have**  $0 < sq\text{-norm}(Mt!i)$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]*  
*M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] sensible-indices*  
**by** (*simp add: M-locale-2*)  
**have**  $(1/sq\text{-norm}(Mt!i)) * ((Mt!i) \cdot (Mt!j)) = (1/sq\text{-norm}(Mt!i)) * sq\text{-norm}(Mt!i)$   
**using** *sensible-indices diag sq-norm-vec-as-cscalar-prod[of Mt!i]* **by** *auto*  
**then have**  $(1/sq\text{-norm}(Mt!i)) * ((Mt!i) \cdot (Mt!j)) = 1$   
**using** *nonzero* **by** *auto*  
**then show** *?thesis* **using** *diag* **by** *argo*  
**next**  
**case** *off: False*  
**have**  $0 < sq\text{-norm}(Mt!i)$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]*  
*M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)] sensible-indices*  
**by** (*simp add: M-locale-2*)  
**then have**  $0 < 1/sq\text{-norm}(Mt!i)$  **by** *simp*  
**moreover have**  $((Mt!i) \cdot (Mt!j)) = 0$   
**using** *gram-schmidt-fs-lin-indpt.orthogonal[of n (RAT) M i j]* *off sensible-indices*  
*M-locale-1 M-locale-2 gram-schmidt-fs-Rn.main-connect*  
**by** *force*  
**ultimately show** *?thesis* **using** *off* **by** *algebra*  
**qed**  
**moreover then have**  $(1/sq\text{-norm}(Mt!i)) * ((Mt!i) \cdot (Mt!j)) = (1_m \ n)\$(i,j)$   
**using** *sensible-indices unfolding one-mat-def* **by** *simp*  
**ultimately show** *?thesis* **by** *presburger*  
**qed**  
**then have**  $inv\text{-Mt} : (?Mt\text{-inv-mat} * ?Mt\text{-mat}) = 1_m \ n$   
**using** *carrier-mat-inv carrier-mat-Mt*  
**by** *fastforce*  
**then have**  $?Mt\text{-mat} * ?Mt\text{-inv-mat} = 1_m \ n$   
**using** *mat-mult-left-right-inverse[of ?Mt-inv-mat n ?Mt-mat]* *carrier-mat-inv carrier-mat-Mt*  
**by** *argo*  
**then have**  $3 : (?Mt\text{-mat} * ?Mt\text{-inv-mat}) \cdot_v v = v$   
**using** *v-carr* **by** *simp*  
**then have**  $4 : v = gs.\text{sumlist} (\text{map } (\lambda j. (1/sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v)) \cdot_v Mt!j)$   
 $[0 ..< n]$   
**using** *v-carr carrier-mat-inv carrier-mat-Mt 1 2* **by** *auto*  
**have**  $(\text{map } (\lambda j. (1/sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v)) \cdot_v Mt!j) [0 ..< n]$   
 $= (\text{map } (\lambda j. (1/sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v)) \cdot_v gs.\text{gso } j) [0 ..< n]$   
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect[of n RAT M]*

**by auto**  
**then have**  $gs.sumlist (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v) \cdot_v Mt!j) [0 ..< n])$   
 $= gs.sumlist (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v) \cdot_v gs.gso j) [0 ..< n])$   
**by argo**  
**then have**  $v = gs.sumlist (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v) \cdot_v gs.gso j) [0 ..< n])$   
**using 4 by argo**  
**then have**  $v\cdot v = gs.sumlist (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v) \cdot_v gs.gso j) [0 ..< n])\cdot$   
 $gs.sumlist (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v) \cdot_v gs.gso j) [0 ..< n])$   
**by simp**  
**then have**  $a:v\cdot v =$   
 $sum-list(map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(gs.gso j \cdot gs.gso j)) [0..<n])$   
**using** *gram-schmidt-fs-lin-indpt.scalar-prod-lincomb-gso*  
*of n RAT M n*  $(\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)) (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v))$   
*M-locale-2*  
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[of n RAT M] **by force**  
**have**  $(map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(gs.gso j \cdot gs.gso j)) [0..<n])$   
 $= (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(Mt!j \cdot Mt!j)) [0..<n])$   
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect*[of n RAT M]  
**by auto**  
**then have**  $b:sum-list (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(gs.gso j \cdot gs.gso j)) [0..<n])$   
 $= sum-list (map (\lambda j. (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(Mt!j \cdot Mt!j)) [0..<n])$   
**by argo**  
**have**  $(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(Mt!j \cdot Mt!j) =$   
 $(v\cdot(Mt!j))^2/(sq-norm (Mt!j))$  **if sensible-indices:0≤j∧j<n for j**  
**proof-**  
**have**  $nonzero:0 < sq-norm(Mt!j)$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[of n RAT M j]  
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[of n (RAT M)] *sensible-indices*  
**by (simp add: M-locale-2)**  
**moreover have**  $(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(Mt!j \cdot Mt!j)$   
 $= (1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*sq-norm(Mt!j)$   
**using** *sq-norm-vec-as-cscalar-prod*[of Mt!j] **by force**  
**moreover have**  $(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*(1/sq-norm(Mt!j))* ((Mt!j)\cdot v)*sq-norm(Mt!j)$

$$= ((Mt!j) \cdot v)^{\wedge 2} * (1 / sq\text{-norm}(Mt!j))^{\wedge 2} * sq\text{-norm}(Mt!j)$$
**by** (*simp add: power2-eq-square*)  
**moreover have**  $((Mt!j) \cdot v)^{\wedge 2} * (1 / sq\text{-norm}(Mt!j))^{\wedge 2} * sq\text{-norm}(Mt!j) =$   
 $((Mt!j) \cdot v)^{\wedge 2} / (sq\text{-norm}(Mt!j))$   
**using** *nonzero*  
**by** (*simp add: divide-divide-eq-left' power2-eq-square*)  
**moreover have**  $(Mt!j) \cdot v = v \cdot (Mt!j)$  **using** *v-carr dim-vecs-in-Mt sensible-indices*  
**by** (*metis carrier-vecI comm-scalar-prod*)  
**ultimately show** *?thesis* **by** *argo*  
**qed**  
**then have**  $(\text{map } (\lambda j. (1 / sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v) * (1 / sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v) * (Mt!j$   
 $\cdot Mt!j)) [0..<n])$   
 $= (\text{map } (\lambda j. (v \cdot (Mt!j))^{\wedge 2} / (sq\text{-norm}(Mt!j))) [0..<n])$  **by** *force*  
**then have** *c:sum-list*  $(\text{map } (\lambda j. (1 / sq\text{-norm}(Mt!j)) * ((Mt!j) \cdot v) * (1 / sq\text{-norm}(Mt!j)) *$   
 $((Mt!j) \cdot v) * (Mt!j \cdot Mt!j)) [0..<n])$   
 $= \text{sum-list } (\text{map } (\lambda j. (v \cdot (Mt!j))^{\wedge 2} / (sq\text{-norm}(Mt!j))) [0..<n])$  **by** *argo*  
**then have**  $v \cdot v = \text{sum-list } (\text{map } (\lambda j. (v \cdot (Mt!j))^{\wedge 2} / (sq\text{-norm}(Mt!j))) [0..<n])$   
**using** *a b c* **by** *argo*  
**moreover have**  $v \cdot v = v \cdot cv$  **by** *force*  
**ultimately show** *?thesis* **using** *sq-norm-vec-as-cscalar-prod[of v] v-carr* **by** *argo*  
**qed**

**lemma** *basis-decay*:

**fixes** *i::nat*  
**fixes** *j::nat*  
**assumes**  $i < n$   
**assumes**  $i + j < n$   
**shows**  $sq\text{-norm}(Mt!i) \leq \alpha^{\wedge j} * sq\text{-norm}(Mt!(i+j))$   
**using** *assms*  
**proof** (*induct j*)  
**case** *0*  
**have**  $\alpha^{\wedge 0} = 1$  **by** *simp*  
**moreover have**  $sq\text{-norm}(Mt!i) = sq\text{-norm}(Mt!(i+0))$  **by** *simp*  
**moreover have**  $0 \leq sq\text{-norm}(Mt!i)$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M i]*  
 $M\text{-locale-2 } M\text{-locale-1 } \text{gram-schmidt-fs-Rn.main-connect}[of n (RAT M)]$   
*assms* **by** *force*  
**moreover have**  $(0::rat) \leq (1::rat)$  **by** *force*  
**ultimately show** *?case* **by** *simp*  
**next**  
**case** (*Suc j*)  
**have**  $(1::rat) \leq \alpha$  **using** *alpha* **by** *fastforce*  
**moreover have**  $n \geq 0$  **by** *simp*  
**ultimately have**  $(1::rat) \leq \alpha^{\wedge j}$  **by** *simp*  
**moreover have**  $sq\text{-norm}(Mt!(i+j)) \leq \alpha * (sq\text{-norm}(Mt!(i+Suc j)))$   
**using** *reduced M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]*  
*Suc.premis*  
**unfolding** *gs.reduced-def gs.weakly-reduced-def*  
**by** *force*

**moreover have**  $0 \leq \text{sq-norm } (Mt!(i+j))$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[of *n RAT M i+j*]  
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[of *n (RAT M)*]  
*Suc.prem*s **by force**  
**ultimately have**  $\alpha^{\wedge j} * \text{sq-norm } (Mt!(i+j)) \leq \alpha^{\wedge j} * \alpha * (\text{sq-norm } (Mt!(i+Suc\ j)))$   
**by simp**  
**moreover have**  $\text{sq-norm}(Mt!i) \leq \alpha^{\wedge j} * \text{sq-norm } (Mt!(i+j))$   
**using** *Suc* **by linarith**  
**ultimately have**  $\text{sq-norm}(Mt!i) \leq \alpha^{\wedge j} * \alpha * (\text{sq-norm } (Mt!(i+Suc\ j)))$  **by order**  
**moreover have**  $\alpha^{\wedge j} * \alpha = \alpha^{\wedge (Suc\ j)}$  **by simp**  
**ultimately show** *?case* **by argo**  
**qed**

**lemma** *basis-decay-cor*:

**fixes** *i::nat*  
**fixes** *j::nat*  
**assumes**  $i < n$   
**assumes**  $j < n$   
**assumes**  $i \leq j$   
**shows**  $\text{sq-norm } (Mt!i) \leq \alpha^{\wedge n} * \text{sq-norm}(Mt!j)$

**proof** –

**have**  $1 : \text{sq-norm } (Mt!i) \leq \alpha^{\wedge (j-i)} * \text{sq-norm}(Mt!j)$   
**using** *basis-decay*[of *i j-i*] *assms*  
**by simp**  
**have**  $\alpha^{\wedge (j-i)} \leq \alpha^{\wedge n}$  **using** *assms* **using** *alpha* **by force**  
**then have**  $\alpha^{\wedge (j-i)} * \text{sq-norm}(Mt!j) \leq \alpha^{\wedge n} * \text{sq-norm}(Mt!j)$   
**using** *mult-right-mono* **by blast**  
**then show** *?thesis* **using** *1* **by order**

**qed**

**theorem** *babai-correct*:

**shows**  $\text{real-of-rat } ((\text{sq-norm } (s\ n))::\text{rat}) \leq (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha^{\wedge n}) * \varepsilon * \text{closest-distance-sq}) \wedge s\ n \in \text{coset}$

**proof** –

**let** *?s = s n*  
**let** *?component = ( $\lambda i. (?s * Mt!i)^2 / (\text{sq-norm } (Mt!i))$ )*  
**obtain** *v* **where** *wit-v:witness v ( $\varepsilon$ -closest)*  
**using** *witness-exists* **by force**  
**have** *split-norm:sq-norm ?s = sum-list (map ?component [0..<n])*  
**using** *s-dim*[of *n*] *sq-norm-from-Mt*[of *?s*] **by fast**  
**have**  $I+1 \in \mathbb{N}$  **using** *I-geq*  
**using** *Nats-0 Nats-1 Nats-add R.add.l-inv-ex R.add.r-inv-ex add-diff-cancel-right'*

*cring-simprules(21) rangeI range-abs-Nats verit-la-disequality verit-minus-simplify(3)*

*zabs-def zle-add1-eq-le* **by auto**

**then obtain** *Inat* **where** *Inat-def:int Inat = I+1*

**using** *Nats-cases* **by metis**

**then have** *Inat-small:Inat ≤ n* **using** *I-leq* **by fastforce**

```

then have  $[0..<n] = [0..<Inat] @ [Inat..<n]$ 
by (metis bot-nat-0.extremum-uniqueI le-Suc-ex nat-le-linear upt-add-eq-append)
then have split-norm-sum:sq-norm  $?s = \text{sum-list } (\text{map } ?\text{component } [0..<Inat])$ 
+  $\text{sum-list } (\text{map } ?\text{component } [Inat..<n])$ 
using split-norm by force

have  $?\text{component } i \leq \varepsilon\text{-closest}$  if  $i:Inat \leq i \wedge i < n$  for  $i$ 
proof–
  have ge0:sq-norm  $(Mt!i) > 0$ 
  using gram-schmidt-fs-lin-indpt.sq-norm-pos[of  $n$  RAT M i]
  M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of  $n$  (RAT M)]
   $i$  by force
then have  $?\text{component } i = (v \cdot Mt!i)^2 / (\text{sq-norm } (Mt!i))$ 
  using ge0 correct-coord[of  $v$   $i$ ] wit-v Inat-def i
  by auto
also have  $(v \cdot Mt!i)^2 \leq (\text{sq-norm } v) * \text{sq-norm } (Mt!i)$ 
  using scalar-prod-Cauchy[of  $v$   $n$   $Mt!i$ ]
  dim-vecs-in-Mt[of  $i$ ] carrier-vecI[of  $v$ ] carrier-vecI[of  $Mt!i$ ] wit-v
   $i$ 
  unfolding witness-def
  by algebra
also have  $\text{sq-norm } v \leq \varepsilon\text{-closest}$ 
  using wit-v unfolding witness-def by fast
finally show ?thesis using ge0
  by (simp add: divide-right-mono)
qed
then have  $\bigwedge x. x \in \text{set } [Inat..<n] \implies ?\text{component } x \leq (\lambda i. \varepsilon\text{-closest}) x$  by simp
then have  $\text{sum-list } (\text{map } ?\text{component } [Inat..<n]) \leq \text{sum-list } (\text{map } (\lambda i. \varepsilon\text{-closest})$ 
 $[Inat..<n])$ 
  using sum-list-mono[of  $[Inat..<n]$   $?\text{component } (\lambda i. \varepsilon\text{-closest})$ ] by argo
then have right-sum:sum-list  $(\text{map } ?\text{component } [Inat..<n]) \leq (\text{rat-of-nat } (n - Inat)) * \varepsilon\text{-closest}$ 
  using sum-list-triv[of  $\varepsilon\text{-closest } [Inat..<n]$ ] by force
have  $(1::\text{rat}) \leq \alpha$  using alpha by fastforce
moreover have  $n \geq 0$  by simp
ultimately have  $(1::\text{rat}) \leq \alpha \hat{n}$  by simp
moreover have  $(0::\text{rat}) \leq 1$  by simp
moreover have  $0 \leq (\text{rat-of-nat } (n - Inat)) * \varepsilon\text{-closest}$ 
proof–
  have  $0 \leq (\text{rat-of-nat } (n - Inat))$  using Inat-small by fast
moreover have  $0 \leq \varepsilon\text{-closest}$ 
proof(cases closest-distance-sq = 0)
  case  $t:\text{True}$ 
  then show ?thesis using  $\varepsilon\text{-closest-lemma}$  closest-distance-sq-pos unfolding
close-condition-def
  by auto
next
case  $f:\text{False}$ 
then show ?thesis using  $\varepsilon\text{-closest-lemma}$  closest-distance-sq-pos unfolding

```

```

close-condition-def
  by (smt (verit, del-insts) zero-le-of-rat-iff)
qed
ultimately show ?thesis by blast
qed
ultimately have (rat-of-nat (n-Inat))*ε-closest ≤ (rat-of-nat (n-Inat))*ε-closest
* αn
  using mult-left-mono[of 1 αn (rat-of-nat (n-Inat))*ε-closest] by linarith
  then have sum-list (map ?component [Inat..n]) ≤ (rat-of-nat (n-Inat))*ε-closest*αn
using right-sum by order
  then have right-sum-alpha:sum-list (map ?component [Inat..n]) ≤ (rat-of-nat
(n-Inat))*αn*ε-closest
  by algebra
  have sum-list (map ?component [0..Inat]) + sum-list (map ?component [Inat..n]) ≤
(rat-of-int n)*αn*ε-closest
  proof (cases Inat = 0)
    case Inat:True
      then have sum-list (map ?component [0..Inat]) = 0 by auto
      then have sum-list (map ?component [0..Inat]) + sum-list (map ?component
[Inat..n]) ≤ (rat-of-int (n-Inat))*αn * ε-closest
        using right-sum-alpha by simp
      also have n-Inat = n using Inat by simp
      finally show ?thesis by linarith
    next
      case False
        then have non-zero:Inat>0 by blast
        then have I-not-min:I≥0 using Inat-def by simp
        then have non-empty:I = Max ({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat
set)
          unfolding I-def by presburger
        then have max:Inat-1 = Max({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat
set)
          using Inat-def by linarith
        then have Inat-1 ∈ ({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat
set)
          proof-
            have finite ({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat set)
              by simp
            moreover have ({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat
set)≠{}
              using I-not-min unfolding I-def by presburger
            ultimately show Inat-1 ∈ ({i∈{0..n}. ((sq-norm (Mt!i)::rat))≤4*ε-closest}::nat
set)
              using max eq-Max-iff by blast
          qed
        then have 2:(sq-norm (Mt!(Inat-1))::rat)≤4*ε-closest by blast
        have (1::rat) ≤α using alpha by fastforce
        moreover have n≥0 by simp
        ultimately have (1::rat)≤αn by simp

```

**then have**  $((1/4)::rat) \leq 1/4 * \alpha^{\wedge n}$  **by** *auto*  
**then have**  $(0::rat) < 1/4 * \alpha^{\wedge n}$  **by** *linarith*  
**moreover have**  $0 < (sq\text{-norm } (Mt!(Inat-1)))::rat$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[of  $n$  *RAT M Inat-1*]  
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[of  $n$  (*RAT M*)]  
*non-zero Inat-small* **by** *force*  
**ultimately have**  $bound:1/4 * \alpha^{\wedge n} * (sq\text{-norm } (Mt!(Inat-1))) \leq ((1/4 * \alpha^{\wedge n}) * 4 * \varepsilon\text{-closest})$   
**using** *2* **by** *auto*  
**have**  $?component\ i \leq \alpha^{\wedge n} * \varepsilon\text{-closest}$  **if** *list1:i < Inat* **for**  $i$   
**proof-**  
**have**  $1:0 < n-i$  **using** *list1 Inat-small* **by** *simp*  
**then have**  $?s.Mt!i = (s\ (n-i)).Mt!i$   
**using** *coord-invariance*[of  $n-i\ n-i\ i$ ] **by** *fastforce*  
**then have**  $abs(?s.Mt!i) \leq (1/2) * (sq\text{-norm } (Mt!i))$   
**using** *small-orth-coord*[of  $n-i$ ] *1* **by** *force*  
**then have**  $(?s.Mt!i)^{\wedge 2} \leq ((1/2) * (sq\text{-norm } (Mt!i)))^{\wedge 2}$   
**by** (*meson abs-ge-self abs-le-square-iff ge-trans*)  
**moreover have**  $ge0:sq\text{-norm } (Mt!i) > 0$   
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[of  $n$  *RAT M i*]  
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[of  $n$  (*RAT M*)]  
*list1 Inat-small* **by** *force*  
**ultimately have**  $?component\ i \leq ((1/2) * (sq\text{-norm } (Mt!i)))^{\wedge 2} / (sq\text{-norm } (Mt!i))$   
**using** *divide-right-mono* **by** *auto*  
**also have**  $((1/2) * (sq\text{-norm } (Mt!i)))^{\wedge 2} / (sq\text{-norm } (Mt!i)) = 1/4 * (sq\text{-norm } (Mt!i))^{\wedge 2} / (sq\text{-norm } (Mt!i))$   
**by** (*metis (no-types, lifting) gs.cring-simprules(12) numeral-Bit0-eq-double power2-eq-square times-divide-eq-left times-divide-times-eq*)  
**also have**  $1/4 * (sq\text{-norm } (Mt!i))^{\wedge 2} / (sq\text{-norm } (Mt!i)) = 1/4 * (sq\text{-norm } (Mt!i))$   
**using** *ge0* **by** (*simp add: power2-eq-square*)  
**also have**  $1/4 * sq\text{-norm } (Mt!i) \leq 1/4 * \alpha^{\wedge n} * (sq\text{-norm } (Mt!(Inat-1)))$   
**using** *basis-decay-cor*[of  $i$  *Inat-1*] *list1 Inat-small mult-left-mono*  
*of sq-norm (Mt!i)  $\alpha^{\wedge n} * (sq\text{-norm } (Mt!(Inat-1)))$   $1/4$*   
**by** *linarith*  
**finally have**  $?component\ i \leq 1/4 * \alpha^{\wedge n} * 4 * \varepsilon\text{-closest}$   
**using** *bound* **by** *linarith*  
**also have**  $1/4 * \alpha^{\wedge n} * 4 * \varepsilon\text{-closest} = \alpha^{\wedge n} * \varepsilon\text{-closest}$  **by** *force*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**then have**  $sum\text{-list } (map\ ?component\ [0..<Inat]) \leq sum\text{-list } (map\ (\lambda i. \alpha^{\wedge n} * \varepsilon\text{-closest})\ [0..<Inat])$   
**using** *sum-list-mono*[of  $[0..<Inat]$  *?component*  $(\lambda i. \alpha^{\wedge n} * \varepsilon\text{-closest})$ ] **by** *fastforce*  
**then have**  $sum\text{-list } (map\ ?component\ [0..<Inat]) \leq (rat\text{-of-int } Inat) * \alpha^{\wedge n} * \varepsilon\text{-closest}$   
**using** *sum-list-triv*[of  $\alpha^{\wedge n} * \varepsilon\text{-closest}$   $[0..<Inat]$ ] **by** *auto*  
**then have**  $(sum\text{-list } (map\ ?component\ [0..<Inat])) + sum\text{-list } (map\ ?component$

```

[Inat..<n])
      ≤ (rat-of-int Inat)*α∧n * ε-closest+(rat-of-int (n-Inat))*α∧n *
ε-closest
  using right-sum-alpha by linarith
  then have (sum-list (map ?component [0..<Inat])) + sum-list (map ?component
[Inat..<n])
      ≤ ((rat-of-int Inat)+(rat-of-int (n-Inat)))*α∧n * ε-closest
  using gs.cring-simprules(13) by auto
  then show ?thesis
  by (metis (no-types, lifting) Inat-small add-diff-inverse-nat diff-is-0-eq' less-nat-zero-code

      of-int-of-nat-eq of-nat-add zero-less-diff)
qed
then have sq-norm ?s ≤ (rat-of-int n)*α∧n * ε-closest
  using split-norm-sum by argo
then have real-of-rat (sq-norm ?s) ≤ real-of-rat ((rat-of-int n)*α∧n * ε-closest)
  by (simp add: of-rat-less-eq)
then have real-of-rat (sq-norm ?s) ≤ real-of-rat ((rat-of-int n)*α∧n) * (real-of-rat
ε-closest)
  using of-rat-mult by metis
moreover have real-of-rat ε-closest ≤ ε * closest-distance-sq
  using ε-closest-lemma closest-distance-sq-pos unfolding close-condition-def by
blast
ultimately show ?thesis
  using coset-s alpha
  using mult-left-mono[of real-of-rat ε-closest ε * closest-distance-sq real-of-rat
(rat-of-int (int n) * α∧n)]
  by simp
qed
end

```

In this section we remove the arbitrary constant  $\epsilon$  and clean up the assumptions and results.

```

locale babai-with-assms = babai +
  fixes mat-M :: rat mat
  assumes basis: lin-indep M
  defines mat-M ≡ mat-of-cols n (RAT M)
  assumes reduced: weakly-reduced M n
  assumes non-trivial: 0 < n
  assumes alpha: α ≥ 4/3
begin

```

```

sublocale vec-space TYPE(rat) n .

```

```

definition mat-M-inv ≡
  (if (invertible-mat mat-M) then SOME B. (inverts-mat B mat-M) ∧ (inverts-mat
mat-M B) else (0m n n))

```

**lemma** *carrier-M*:  
**shows**  $\text{mat-}M \in \text{carrier-mat } n \ n$   
**using** *basis unfolding gs.lin-indpt-list-def mat-M-def* **by** *auto*

**lemma** *mat-M-inv-is-inv*:  
**shows** *invertible-mat mat-M*

**proof** –

**have**  $\text{vec-space.rank } n \ \text{mat-}M = n$   
**using** *vec-space.lin-indpt-full-rank[of mat-M n n] carrier-M basis mat-M-def*  
**unfolding** *cof-vec-space.lin-indpt-list-def*  
**by** *auto*  
**then have**  $\det \text{mat-}M \neq 0$  **using** *vec-space.det-rank-iff[of mat-M n] carrier-M*  
**by** *fastforce*  
**then show** *invertible-mat mat-M* **using** *invertible-det[of mat-M n] carrier-M* **by**  
*fastforce*  
**qed**

**abbreviation** *babai-out* **where**  $\text{babai-out} \equiv \text{babai-of-LLL } (-t) \ (\text{RAT } M)$

**lemma** *babai-with-assms-ε-connect*:

**shows**  $\text{babai-with-assms-}\varepsilon \ M \ t \ \alpha \ 2$   
**using** *mat-M-inv-is-inv unfolding babai-with-assms-ε-def babai-with-assms-ε-axioms-def*  
**using** *babai-axioms alpha basis mat-M-def non-trivial reduced* **by** *simp*

This shows that the output, which is of the form v-t, with v in L, is short

**lemma** *babai-correct*:

**shows**  $\text{real-of-rat } (\text{sq-norm } (\text{babai-out})) \leq (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \widehat{n}) * \text{closest-distance-sq}) \wedge (\text{babai-out}) \in \text{coset}$

**proof** –

**have**  $*: \text{real-of-rat } (\text{sq-norm } (\text{babai-out})) \leq (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \widehat{n}) * \varepsilon * \text{closest-distance-sq})$  **if**  $\text{eps:}1 < \varepsilon$  **for**  $\varepsilon$

**proof** –

**have**  $\text{babai-with-assms-}\varepsilon \ M \ t \ \alpha \ \varepsilon$   
**using** *eps mat-M-inv-is-inv unfolding babai-with-assms-ε-def babai-with-assms-ε-axioms-def*  
**using** *babai-axioms alpha basis mat-M-def non-trivial reduced* **by** *blast*  
**then show**  $?thesis$  **using**  $\text{babai-with-assms-}\varepsilon.\text{babai-correct}[of \ M \ t \ \alpha \ \varepsilon]$   $\text{babai-with-assms-}\varepsilon.\text{babai-to-help}[of \ M \ t \ \alpha \ \varepsilon]$   
*s-def* **by** *simp*

**qed**

**have**  $\text{real-of-rat } (\text{sq-norm } (\text{babai-out})) \leq (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \widehat{n}) * \text{closest-distance-sq})$

**proof**(*rule ccontr*)

**assume**  $\neg ?thesis$

**then have**  $\text{not}:(\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \widehat{n}) * \text{closest-distance-sq}) < \text{real-of-rat } (\text{sq-norm } (\text{babai-of-LLL } (-t) \ (\text{RAT } M)))$  **by** *auto*

**then obtain**  $\varepsilon$  **where**  $(\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \widehat{n}) * \varepsilon * \text{closest-distance-sq}) < \text{real-of-rat } (\text{sq-norm } (\text{babai-of-LLL } (-t) \ (\text{RAT } M))) \wedge 1 < \varepsilon$

**proof**(*cases closest-distance-sq = 0*)

**case** *True*

**then have**  $(\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * 2 * \text{closest-distance-sq}) =$   
 $(\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq})$  **by auto**  
**moreover have**  $1 < (2::\text{real})$  **by simp**  
**ultimately show** *?thesis* **using** *that[of 2]* **not by presburger**  
**next**  
**case** *f:False*  
**then have**  $\text{pos}: 0 < (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq})$   
**using** *babai-with-assms-ε.closest-distance-sq-pos[of M t α 11/10]*  
**using** *mat-M-inv-is-inv unfolding babai-with-assms-ε-def babai-with-assms-ε-axioms-def*  
**using** *babai-axioms alpha basis mat-M-def non-trivial reduced by simp*  
**obtain** *delta* **where**  $\text{delta}: (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq})$   
 $< \text{delta} \wedge \text{delta} < \text{real-of-rat } (\text{sq-norm } (\text{babai-of-LLL } (-t) (\text{RAT } M)))$   
**using** *dense not by blast*  
**define**  $\varepsilon$  **where**  $e: \varepsilon = \text{delta} / (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq})$   
**then have**  $1 < \varepsilon$   
**using** *pos delta divide-strict-right-mono*  
 $\text{divide-eq-1-iff}[of (\text{real-of-rat } (\text{rat-of-int } (\text{int } n) * \alpha \hat{n}) * \text{closest-distance-sq})$   
 $(\text{real-of-rat } (\text{rat-of-int } (\text{int } n) * \alpha \hat{n}) * \text{closest-distance-sq})]$   
**by auto**  
**moreover have**  $\varepsilon * (\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq}) =$   
 $\text{delta}$  **using** *e pos*  
**by** *(metis nonzero-divide-eq-eq rel-simps(70))*  
**ultimately show** *?thesis* **using** *that[of ε]* **not delta by argo**  
**qed**  
**then show** *False* **using** *\*[of ε]* **by linarith**  
**qed**  
**then show** *?thesis*  
**using** *babai-with-assms-ε.babai-correct babai-with-assms-ε-connect babai-with-assms-ε.babai-to-help[of*  
 $M t \alpha 2]$   
*s-def* **by force**  
**qed**

This shifts the output, which is of the form  $v - t$ , with  $v \in L$ , back to  $v$ .

**abbreviation**  $vL$  **where**  $vL \equiv \text{map-vec int-of-rat } (\text{babai-out } + t)$

**lemma** *shifted-babai-correct*:

**shows**  $vL \in L$

$\wedge \text{real-of-rat } (\text{sq-norm } (\text{map-vec rat-of-int } (vL) - t)) \leq$   
 $\text{real-of-rat } ((\text{rat-of-int } n) * \alpha \hat{n}) * \text{closest-distance-sq}$

**proof** –

**let**  $?vC = (\text{babai-of-LLL } (-t) (\text{RAT } M))$

**have**  $t\text{-carr}: t \in \text{carrier-vec } n$  **using** *babai-axioms* **unfolding** *babai-def* **by fast-force**

**have**  $?vC \in \text{coset}$  **using** *babai-correct* **by blast**

**then obtain**  $v$  **where**  $v: ?vC = \text{of-int-hom.vec-hom } v - t \wedge v \in L$  **by blast**

**then have**  $?vC + t = \text{of-int-hom.vec-hom } v + 0_v n$  **using** *t-carr* **by force**

**then have**  $2: ?vC + t = \text{of-int-hom.vec-hom } v$  **using** *babai-with-assms-ε.lattice-carrier*  
 $v$  *babai-with-assms-ε-connect* **by force**

```

then have 1:map-vec int-of-rat (?vC + t) = map-vec int-of-rat (of-int-hom.vec-hom
v) by fastforce
have (map-vec int-of-rat (of-int-hom.vec-hom v))$i = v$i if i:i<n for i
using i babai-with-assms-ε.lattice-carrier v babai-with-assms-ε-connect by fast-
force
then have map-vec int-of-rat (of-int-hom.vec-hom v) = v using babai-with-assms-ε.lattice-carrier
v babai-with-assms-ε-connect by auto
then have part1:map-vec int-of-rat (?vC + t) ∈ L using 1 v
by auto
have map-vec rat-of-int (map-vec int-of-rat (?vC + t)) = (?vC + t) using 1 2
by fastforce
then have map-vec rat-of-int (map-vec int-of-rat (?vC + t)) - t = ?vC
using t-carr part1 babai-with-assms-ε.lattice-carrier v babai-with-assms-ε-connect
by auto
then show ?thesis using babai-correct part1
by presburger
qed
end

```

Here we prove correctness of the full algorithm, which starts with a non-reduced basis and outputs a vector in L close to t.

```

locale full-babai-with-assms =
fixes target::rat vec
fixes n::nat
fixes fs-init::int vec list
fixes α::rat
assumes non-triv:0<n
assumes t-dim:dim-vec target = n
assumes LLL-assms:LLL.LLL-with-assms n n fs-init α
begin

sublocale vec-space TYPE(rat) n .

abbreviation B::real where B ≡ (real-of-rat (α ^ n) * (n)) * babai.closest-distance-sq
fs-init target
abbreviation out where out ≡ (full-babai fs-init target α)

lemma LLL-output-babai-assms:
shows babai-with-assms (map of-int-hom.vec-hom (LLL-Impl.reduce-basis α fs-init))
target α
proof –
define fs where fs = (LLL-Impl.reduce-basis α fs-init)
have 1:LLL.reduced n α fs n ∧ LLL.lin-indep n fs ∧ length(fs) = n
using LLL-with-assms.reduce-basis[of n n fs-init α fs] LLL-assms unfolding
fs-def by blast
then have 2:LLL.weakly-reduced n α fs n unfolding gram-schmidt-fs.reduced-def
by simp
moreover have babai (map of-int-hom.vec-hom (reduce-basis α fs-init)) target

```

**using 1 unfolding** *babai-def fs-def t-dim* **by** *fastforce*  
**moreover have** *gram-schmidt-fs.weakly-reduced n*  
*(map of-int-hom.vec-hom (map of-int-hom.vec-hom fs)) α*  
*n*  
**using 2 by force**  
**moreover have**  $4/3 \leq \alpha$  **using** *LLL-assms* **unfolding** *LLL-with-assms-def* **by**  
*simp*  
**ultimately show** *?thesis*  
**using** *LLL-assms 1 t-dim non-triv*  
**unfolding** *babai-with-assms-def babai-with-assms-axioms-def fs-def* **by** *auto*  
**qed**

**lemma** *full-babai-correct:*

**shows** *real-of-rat (sq-norm ((map-vec rat-of-int out) - target))*  
 $\leq$  *real-of-rat (α) ^ n \* real n \* babai.closest-distance-sq fs-init target ∧*  
*out ∈ vec-module.lattice-of n fs-init*

**proof** –

**define** *fs* **where** *fs = (LLL-Impl.reduce-basis α fs-init)*  
**have** *bab:babai-with-assms fs target α*  
**using** *LLL-output-babai-assms* **unfolding** *fs-def* **by** *simp*  
**then have** *l:length fs = n*  
**unfolding** *babai-with-assms-def babai-with-assms-axioms-def babai-def*  
**using** *t-dim* **by** *simp*  
**have** *out = map-vec int-of-rat*  
*(babai-of-LLL*  
*(- target)*  
*(LLL.RAT fs)*  
*+ target)*  
**unfolding** *fs-def* **using** *full-babai.simps[of fs-init target α]* **by** *argo*  
**then have** *out = map-vec int-of-rat*  
*(babai-of-LLL (uminus target) (LLL.RAT fs)*  
*+ target)*  
**using** *babai-with-assms.babai-with-assms-ε-connect*  
*babai-with-assms-ε.babai-to-help[of fs target] bab* **by** *metis*  
**then have** *out ∈ vec-module.lattice-of n fs ∧*  
*real-of-rat*  
 $\|of-int-hom.vec-hom out - target\|^2$   
 $\leq$  *real-of-rat (rat-of-int (int n) \* α ^ n) \**  
*babai.closest-distance-sq fs target*  
**using** *babai-with-assms.shifted-babai-correct[of fs target α] bab l*  
**unfolding** *LLL.L-def*  
**by** *algebra*  
**moreover have** *lattices:vec-module.lattice-of n fs = vec-module.lattice-of n fs-init*  
**using** *LLL-with-assms.reduce-basis(1)[of n n fs-init α fs] LLL-assms l* **unfold-**  
**ing** *fs-def LLL.L-def*  
**by** *argo*  
**moreover have** *babai.closest-distance-sq fs target = babai.closest-distance-sq*  
*fs-init target*

```

proof–
  have  $lf.length\ fs-init = n$ 
    using LLL-assms unfolding LLL-with-assms-def by blast
  then have  $babai\ fs\ target \wedge babai\ fs-init\ target$ 
    using t-dim l LLL-assms unfolding babai-def by argo
  then show ?thesis
    using babai.closest-distance-sq-def[of fs target]
      babai.closest-distance-sq-def[of fs-init target]
      lattices l lf
    unfolding LLL.L-def
    by presburger
  qed
  moreover have  $real-of-rat\ (rat-of-int\ (int\ n) * \alpha \wedge n) = real-of-rat\ (\alpha \wedge n * real\ n$ 
real n
    by (simp add: of-rat-divide of-rat-mult of-rat-power)
  ultimately show ?thesis by force
qed
end

```

Here we prove correctness of the full algorithm, starting with a target vector of TYPE(int).

**lemma** *full-babai-correct-int-target*:

```

assumes full-babai-with-assms (map-vec rat-of-int target) n fs-init  $\alpha$ 
shows  $real-of-int\ (sq-norm\ ((full-babai\ fs-init\ (map-vec\ rat-of-int\ target)\ \alpha) - target))$ 
   $\leq (real-of-rat(\alpha) \wedge n * (n)) * babai.closest-distance-sq\ fs-init\ (map-vec\ rat-of-int\ target) \wedge$ 
   $(full-babai\ fs-init\ (map-vec\ rat-of-int\ target)\ \alpha) \in vec-module.lattice-of\ n\ fs-init$ 

```

**proof**–

```

let  $?t = map-vec\ rat-of-int\ target$ 
have  $real-of-rat\ (sq-norm\ (of-int-hom.vec-hom\ (full-babai\ fs-init\ ?t\ \alpha) - ?t))$ 
   $\leq (real-of-rat\ (\alpha) \wedge n * (n)) * babai.closest-distance-sq\ fs-init\ ?t \wedge$ 
   $(full-babai\ fs-init\ ?t\ \alpha) \in vec-module.lattice-of\ n\ fs-init$ 
  using full-babai-with-assms.full-babai-correct[of ?t n fs-init] assms by blast
moreover have  $real-of-rat\ (sq-norm\ (of-int-hom.vec-hom\ (full-babai\ fs-init\ ?t\ \alpha) - ?t))$ 
   $= real-of-int\ (sq-norm\ ((full-babai\ fs-init\ ?t\ \alpha) - target))$ 

```

**proof**–

```

have  $of-int-hom.vec-hom\ (full-babai\ fs-init\ ?t\ \alpha) - ?t = map-vec\ rat-of-int\ ((full-babai\ fs-init\ ?t\ \alpha) - target)$ 
  by fastforce
moreover have  $sq-norm\ (map-vec\ rat-of-int\ ((full-babai\ fs-init\ ?t\ \alpha) - target))$ 
   $= rat-of-int\ (sq-norm\ ((full-babai\ fs-init\ ?t\ \alpha) - target))$ 
  using sq-norm-of-int[of ((full-babai fs-init ?t alpha) - target)] by blast
ultimately have  $real-of-rat\ (sq-norm\ (of-int-hom.vec-hom\ (full-babai\ fs-init\ ?t\ \alpha) - ?t)) = real-of-rat\ (rat-of-int\ (sq-norm\ ((full-babai\ fs-init\ ?t\ \alpha) - target)))$ 
  by force

```

**also have**  $\text{real-of-rat } (\text{rat-of-int } (\text{sq-norm } ((\text{full-babai fs-init } ?t \alpha) - \text{target})))$   
 $= \text{real-of-int } (\text{sq-norm } ((\text{full-babai fs-init } ?t \alpha) - \text{target}))$  **by simp**  
**finally show**  $?thesis$  .  
**qed**  
**ultimately show**  $?thesis$  **by auto**  
**qed**

The literature typically states the main result using  $2^n$  as the approximation constant, rather than  $\alpha^n * n$ . Here we show that  $\alpha^n * n$  is stronger for  $\alpha = 4/3$ .

**lemma** *clean-bound*:

**fixes**  $n::\text{nat}$   
**shows**  $(4/3)^{\wedge}n * n \leq 2^{\wedge}n$   
**proof**(*induct n*)  
**case** 0  
**then show**  $?case$  **by simp**  
**next**  
**case** (*Suc n*)  
**let**  $?SN = \text{Suc } n$   
**have**  $?SN=1 \vee ?SN=2 \vee 2 < ?SN$  **by fastforce**  
**then show**  $?case$   
**proof**(*elim disjE*)  
**{ assume**  $1: ?SN = 1$   
**then have**  $\text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3)^{\wedge} ?SN) = \text{real-of-rat } ((\text{rat-of-int } 1) * 4/3)$   
**by auto**  
**then show**  $?thesis$  **using 1 by simp}**  
**next**  
**{ assume**  $2: ?SN=2$   
**then have**  $\text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3)^{\wedge} ?SN) = \text{real-of-rat } ((\text{rat-of-int } 2) * (4/3)^{\wedge} 2)$   
**by (metis int-ops(3))**  
**then show**  $?thesis$   
**using 2 by auto}**  
**next**  
**{ assume**  $\text{ind}: ?SN > 2$   
**then have**  $n > 0$  **by simp**  
**then have**  $1: ?SN = n * (?SN/n)$  **by auto**  
**moreover have**  $2: ((4::\text{rat})/3)^{\wedge} ?SN = (4/3)^{\wedge} n * (4/3)^{\wedge} (?SN/n)$  **by auto**  
**ultimately have**  $3: \text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3)^{\wedge} ?SN) = (n * (?SN/n)) * \text{real-of-rat } ((4/3)^{\wedge} n * (4/3)^{\wedge} (?SN/n))$   
**by (metis of-int-of-nat-eq of-rat-mult of-rat-of-nat-eq)**  
**also have**  $(n * (?SN/n)) * \text{real-of-rat } ((4/3)^{\wedge} n * (4/3)^{\wedge} (?SN/n)) = \text{real-of-rat } ((\text{rat-of-int } n) * (4/3)^{\wedge} n * ((?SN/n) * (\text{real-of-rat } (4/3))))$   
**by (simp add: ‹real (Suc n) = real n \* (real (Suc n) / real n)› mult-of-int-commute of-rat-divide of-rat-mult)**  
**finally have**  $\text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3)^{\wedge} ?SN) = \text{real-of-rat } ((\text{rat-of-int } n) * (4/3)^{\wedge} n * ((?SN/n) * (\text{real-of-rat } (4/3))))$   
**by presburger**

**then have**  $\text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3) \wedge ?SN) = \text{real-of-rat } ((\text{rat-of-int } n) * (4/3) \wedge n) * ((?SN/n) * (\text{real-of-rat } (4/3)))$   
**by** *argo*  
**moreover have**  $((?SN/n) * (\text{real-of-rat } (4/3))) \leq 2$   
**proof**–  
**have**  $N\text{-big}: 2 \leq n$  **using** *ind*  
**by** *force*  
**then have**  $4 \leq 2 * n$  **by** *fastforce*  
**then have**  $4 * n + 4 \leq 6 * n$  **by** *fastforce*  
**then have**  $4/3 * (\text{Suc } n) \leq 2 * n$  **by** *auto*  
**moreover have**  $0 < 1/n$  **using**  $N\text{-big}$  **by** *simp*  
**ultimately have**  $(4/3 * ?SN) * (1/n) \leq 2 * n * (1/n)$   
**using**  $N\text{-big mult-right-mono}$ [of  $(4/3 * ?SN) 2 * n (1/n)$ ] **by** *linarith*  
**then have**  $(4/3 * ?SN)/n \leq 2 * n/n$  **by** *argo*  
**then have**  $4/3 * (?SN / n) \leq 2 * (n/n)$  **by** *linarith*  
**then have**  $4/3 * (?SN/n) \leq 2$  **using**  $N\text{-big}$  **by** *auto*  
**moreover have**  $4/3 = \text{real-of-rat } (4/3)$  **using** *of-rat-divide*  
**by** *(metis of-rat-numeral-eq)*  
**ultimately have**  $(\text{real-of-rat } (4/3)) * (?SN/n) \leq 2$  **by** *algebra*  
**then show** *?thesis* **by** *argo*  
**qed**  
**moreover have**  
 $0 \leq \text{real-of-rat } (\text{rat-of-int } (\text{int } n) * (4/3) \wedge n)$  **by** *force*  
**moreover have**  $0 \leq (?SN/n) * (\text{real-of-rat } (4/3))$  **by** *simp*  
**moreover have**  $(4/3) \wedge n * \text{real } n * (\text{real } (\text{Suc } n) / \text{real } n * \text{real-of-rat } (4/3)) = \text{real-of-rat } (\text{rat-of-int } (\text{int } n) * (4/3) \wedge n) * (\text{real } (\text{Suc } n) / \text{real } n * \text{real-of-rat } (4/3))$   
**by** *(simp add: of-rat-divide of-rat-mult of-rat-power)*  
**ultimately have**  $\text{real-of-rat } ((\text{rat-of-int } ?SN) * (4/3) \wedge ?SN) \leq 2 \wedge n * 2$   
**using**  $\text{Suc mult-mono}$ [of  
 $(4/3) \wedge n * \text{real } n$   
 $2 \wedge n$   
 $((?SN/n) * (\text{real-of-rat } (4/3)))$   
 $2]$  **by** *force*  
**then show** *?thesis*  
**by** *(metis 1 3 mult-of-nat-commute of-rat-divide of-rat-numeral-eq of-rat-power power-Suc2)*  
**}**  
**qed**  
**qed**  
**end**

## References

- [1] R. Bottesch, J. Divasón, and R. Thiemann. Two algorithms based on modular arithmetic: lattice basis reduction and Hermite normal form computation. *Archive of Formal Proofs*, March

2021. [https://isa-afp.org/entries/Modular\\_arithmetic\\_LLL\\_and\\_HNF\\_algorithms.html](https://isa-afp.org/entries/Modular_arithmetic_LLL_and_HNF_algorithms.html), Formal proof development.
- [2] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A Verified Factorization Algorithm for Integer Polynomials with Polynomial Complexity. *Archive of Formal Proofs*, February 2018. [https://isa-afp.org/entries/LLL\\_Factorization.html](https://isa-afp.org/entries/LLL_Factorization.html), Formal proof development.
- [3] K. Kreuzer. Hardness of Lattice Problems. *Archive of Formal Proofs*, February 2023. [https://isa-afp.org/entries/CVP\\_Hardness.html](https://isa-afp.org/entries/CVP_Hardness.html), Formal proof development.
- [4] N. Stephens Davidowitz. Lecture 5: CVP and Babais Algorithm, August 2016.